

Modell megvalósítása

Gregorics Tibor

gt@inf.elte.hu

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

Modell megvalósítása

1.rész

Egy feladat elemzése

Gregorics Tibor

gt@inf.elte.hu

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

Feladat: Kurzusok

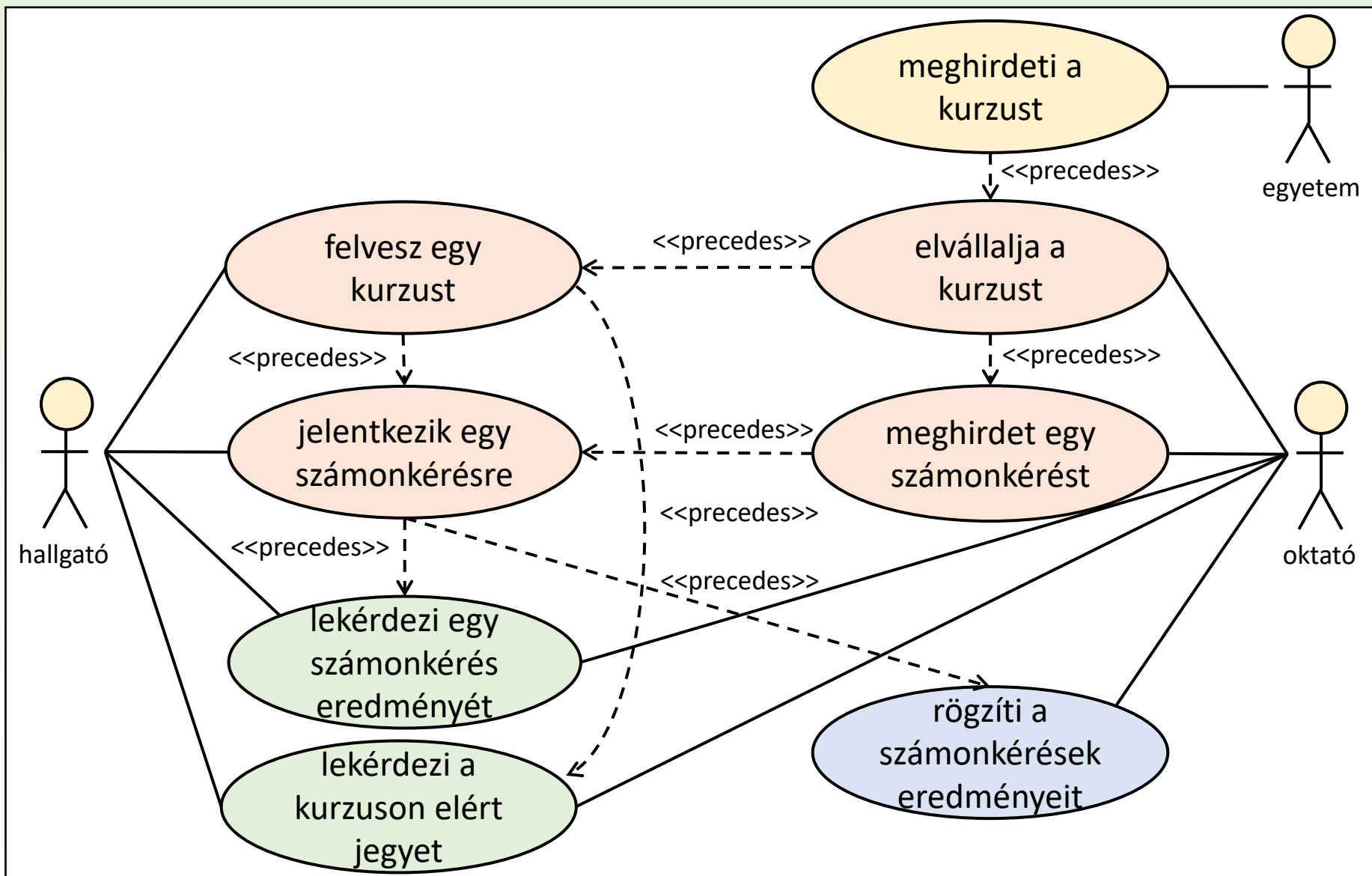
Egy egyetem kurzusokat hirdet meg, amelyeket legfeljebb két oktató tart. A hallgatók jelentkezhetnek a már oktatókkal rendelkező kurzusokra, ha van még ott szabad hely.

Egy kurzus oktatói vizsgákat vagy zárthelyiket tűzhetnek ki a kurzushoz annak jellegétől (előadás vagy gyakorlat) függően. Egy vizsgát a dátuma azonosít, egy zárthelyit a dátuma mellett a neve is. (Az azonos nevű zárthelyik között a későbbi dátumúak javító/pót zárthelyiknek számítanak.) Egy számonkérésre csak az adott kurzus hallgatója iratkozhat fel, ha van még azon szabad hely. Egy hallgató egy előadás legfeljebb három vizsgáján vehet részt (1 vizsga +2 utóvizsga).

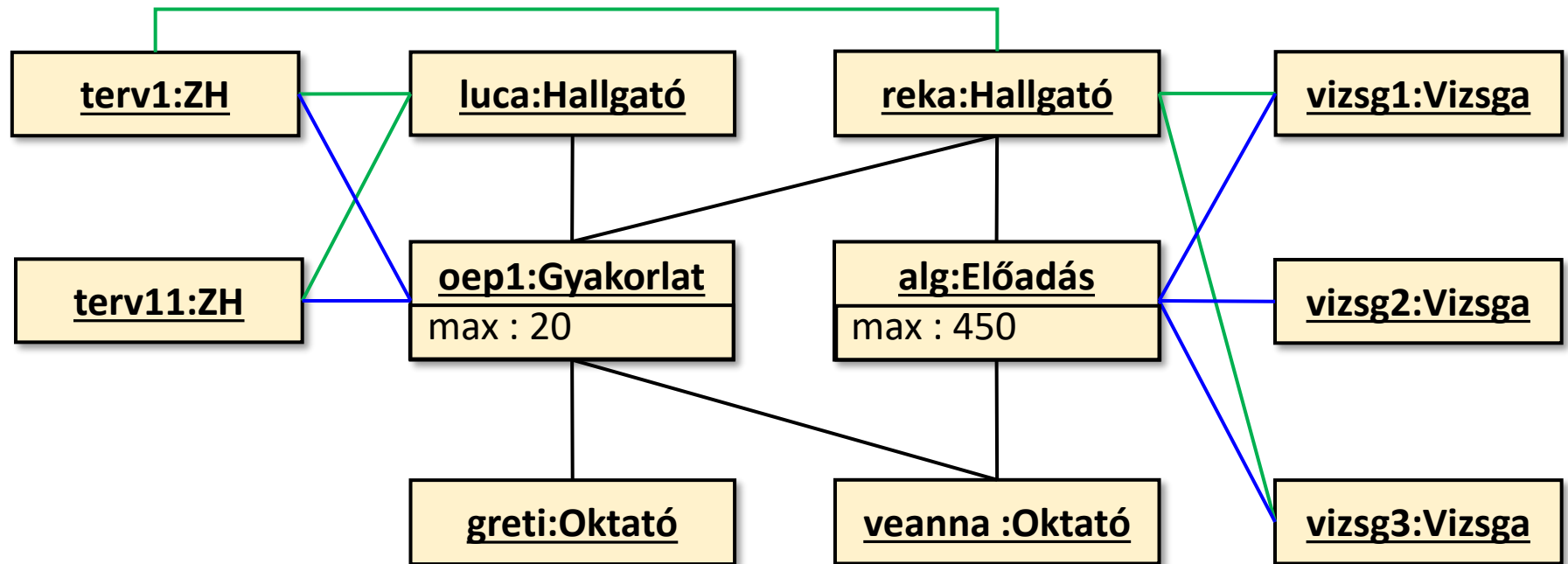
Az oktatók értékelik a számonkéréseket, és ezek alapján a hallgató jegyet kap a kurzusra. Előadásnál a vizsgajegy a legutolsó vizsgán elért eredmény; a gyakorlati jegy a zárthelyik jegyeinek átlaga, de úgy, hogy az ugyanolyan nevű zárthelyik közül mindig csak a legjobb számít bele az átlagba.

A hallgató megtekintheti a számonkéréseinek eredményét, és lekérdezheti a kurzuson elért jegyét. A kurzus oktatói is láthatják a kurzus hallgatóinak eredményeit és jegyét.

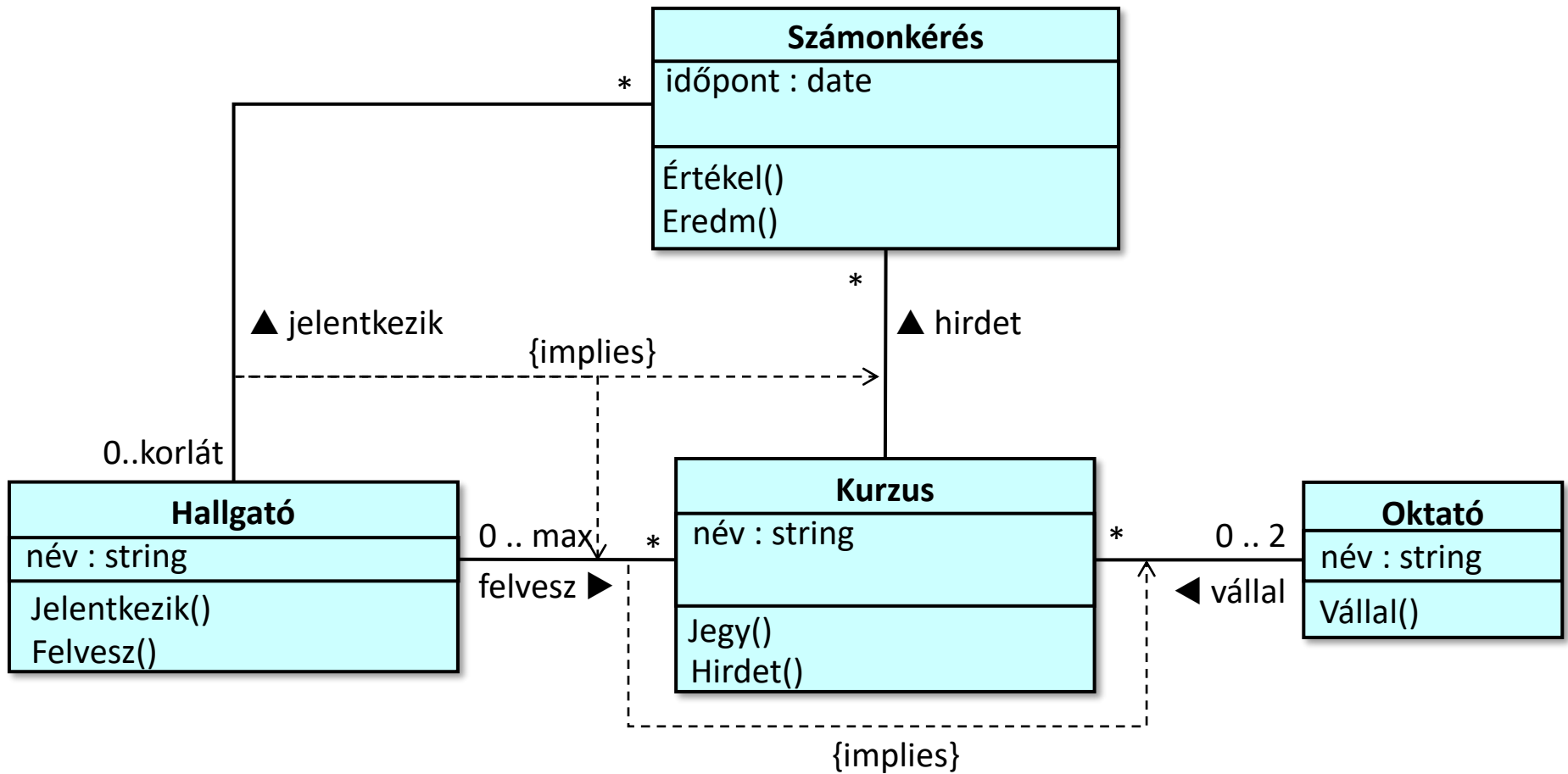
Felhasználói esetek



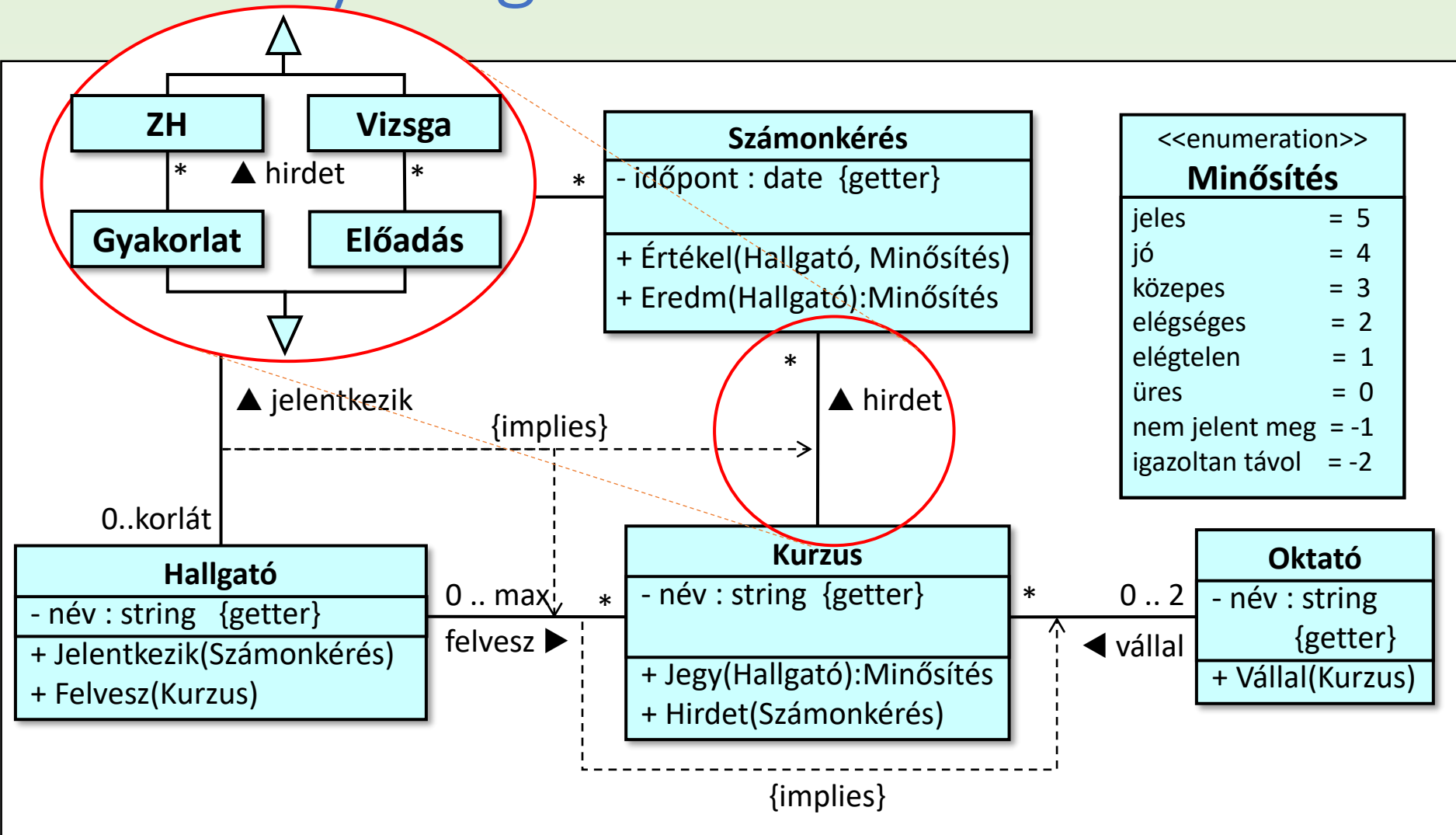
Objektumok és kapcsolataik



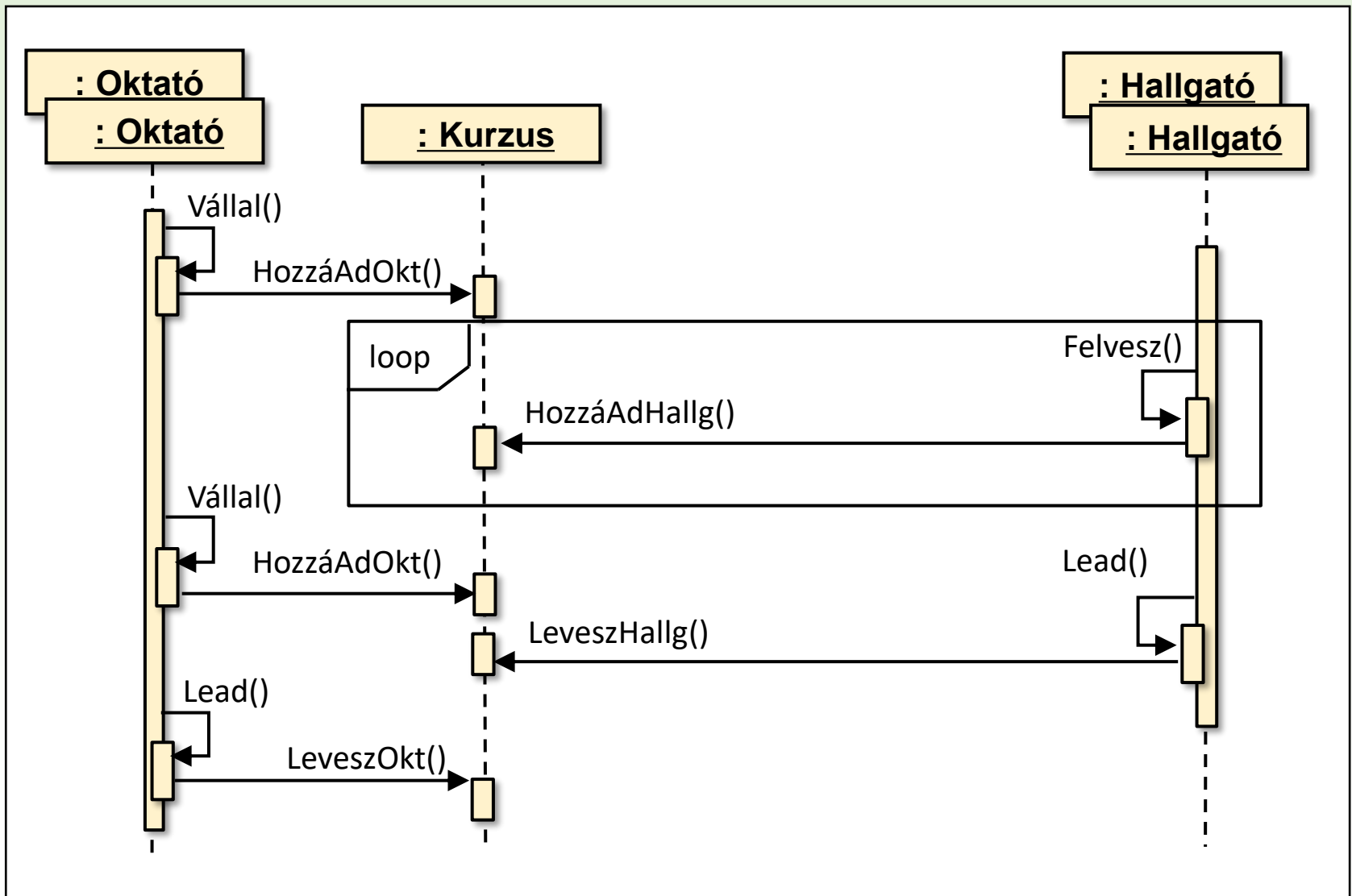
Osztály diagram – elemzési szint



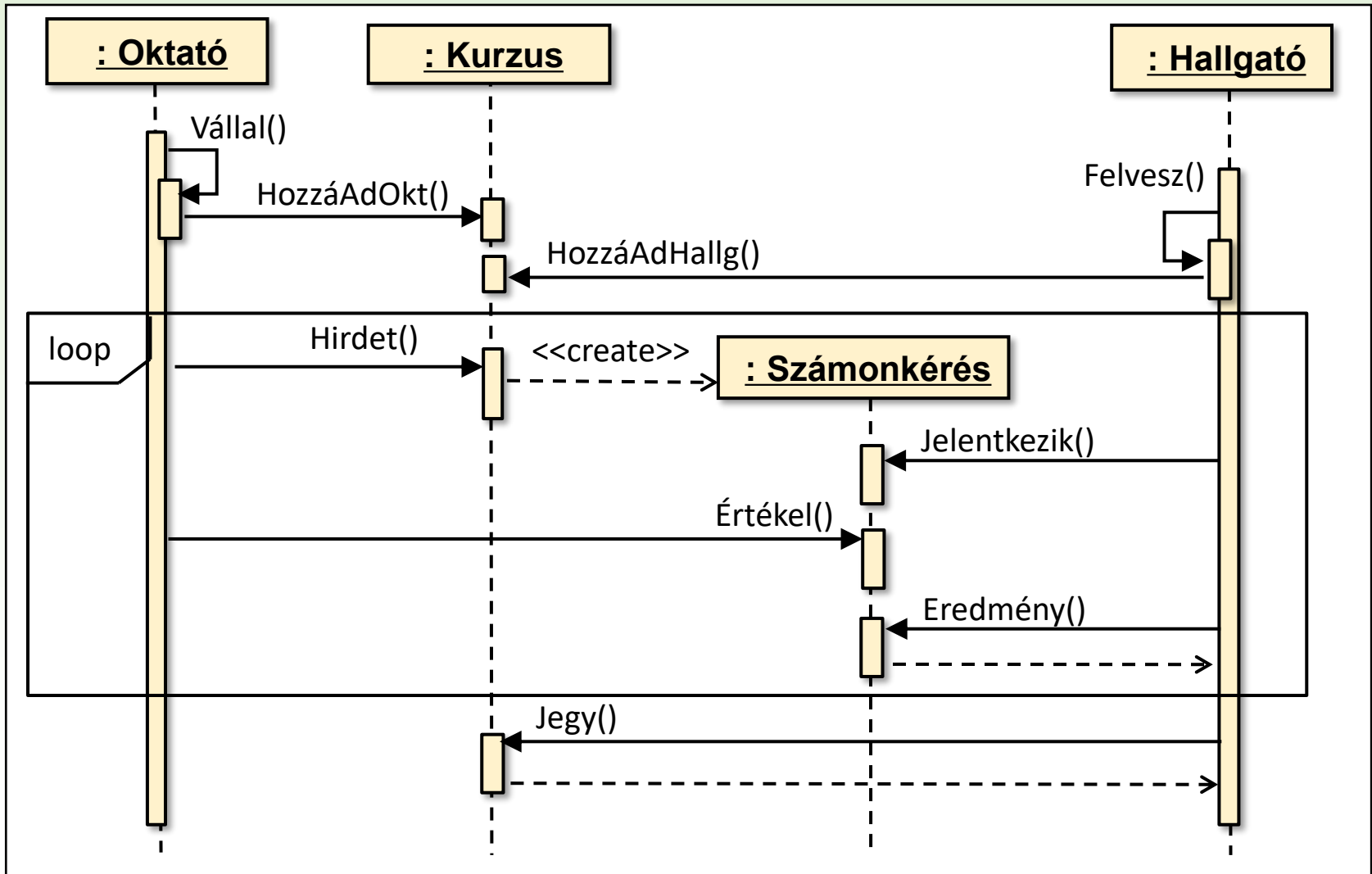
Osztály diagram részletezése



Csatlakozás egy kurzushoz



Számonkérések kezelése



Modell megvalósítása

2.rész

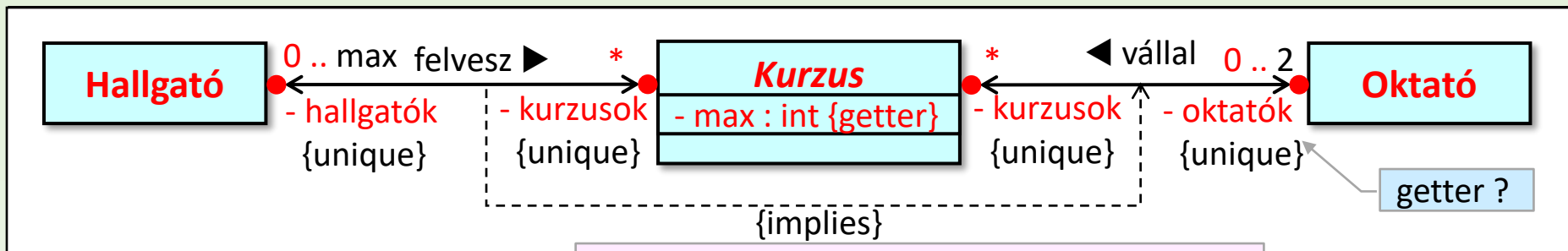
Hallgató-kurzus-oktató
kapcsolatok

Gregorics Tibor

gt@inf.elte.hu

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

Hallgatók, oktatók, kurzusok osztályai



class Course

```
{
    public int Max { get; private set; }
    private readonly List<Student> students = new (); // 0 .. max
    private readonly List<Teacher> teachers = new (); // 0 .. 2
    public Course(int max) { Max = max; }
    ...
}
```

private int max;
public int Max { get { return max; } }
(Lehetne public readonly int Max; is,
ha csak a konstruktor írhatja.)

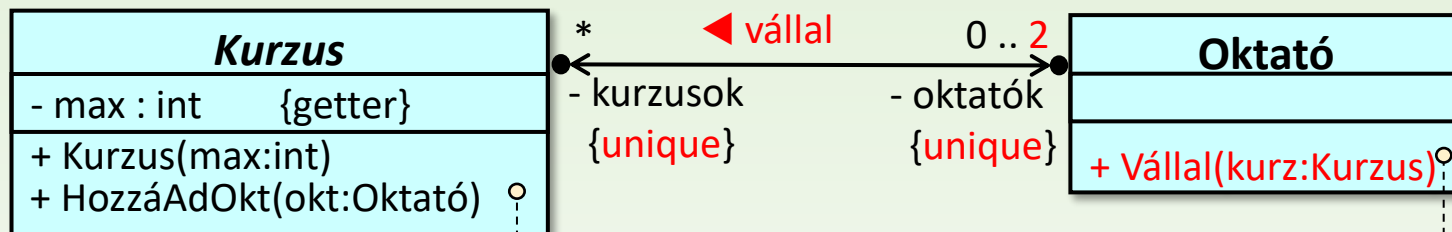
... **class Student**

```
{
    private readonly List<Course> courses = new ();
    ...
}
```

class Teacher

```
{
    private readonly List<Course> courses = new ();
    ...
}
```

Oktató-kurzus kapcsolatok építése



a **multiplicitás** felső korlátjának vizsgálata

```
if |oktatók| = 2 then error endif
if okt in oktatók then return error endif
oktatók.Add(okt)
```

mindkét **unique** feltételt ugyanaz az eldöntés (lineáris keresés) ellenőrzi

```
kurz.HozzáAdOkt(this)
kurzusok.Add(kurz)
```

- a kurz.oktatók nem érhető el közvetlenül
- egy oktató (this) hozzáadása adjon hibát, ha az oktató nem rendelhető a kurzushoz, mert vagy már van ott két oktató, vagy már hozzá van rendelve.

C# kód



```
private readonly List<Teacher> teachers = new (); // 0 .. 2
```

```
public void AddTeacher(Teacher teacher)
{
    if (teachers.Count == 2) throw new TooManyTeachersLeadThisCourse();
    if (teachers.Contains(teacher)) throw new TeacherHasLeadThisCourse();
    teachers.Add(teacher);
}
```

okt in oktatók

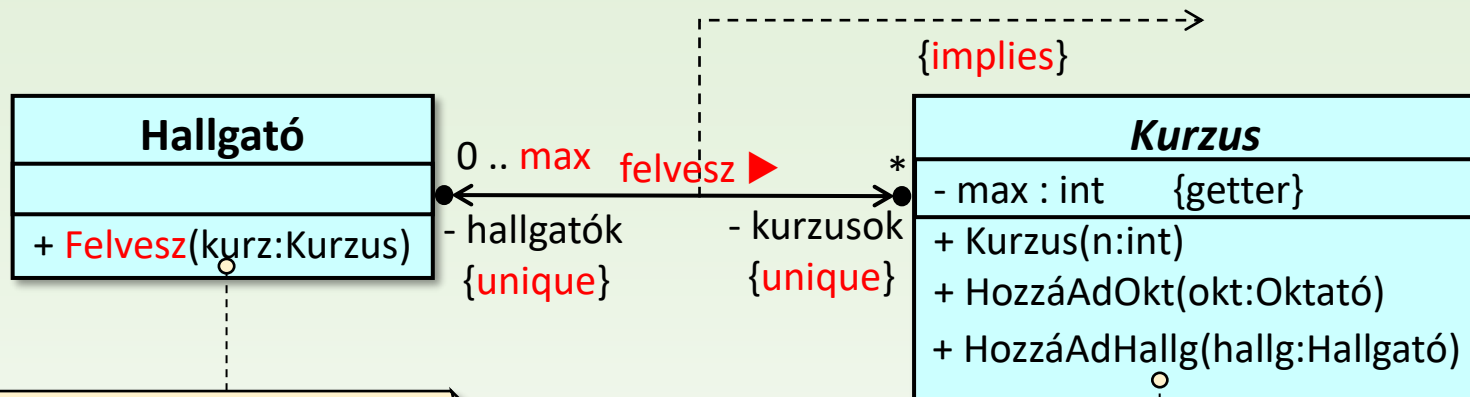
class Course

```
private readonly List<Course> courses = new ();
```

```
public void Undertakes(Course course)
{
    course.AddTeacher(this);
    courses.Add(course);
}
```

class Teacher

Hallgató-kurzus kapcsolatok építése



kurz.HozzáAdHallg(this)
kurzusok.Add(kurz)

- a kurz.hallgatók nem érhető el közvetlenül
- egy hallgató hozzáadása adjon hibát, ha a hallgató (this) nem veheti fel a kurzust, mert vagy nincs szabad hely (**max**), vagy **már felvette**.

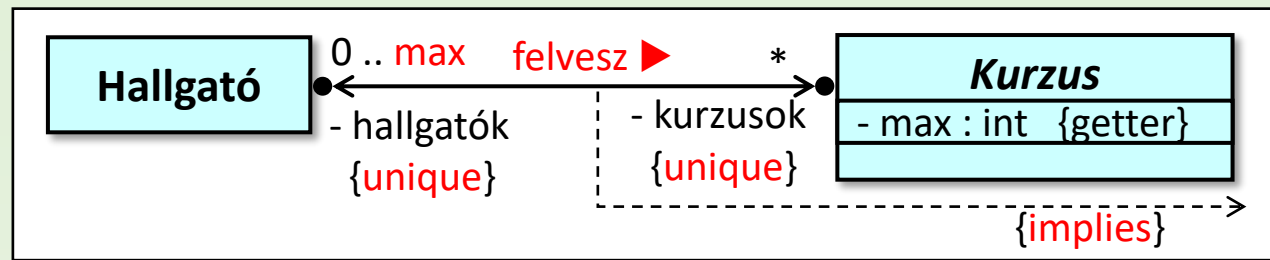
az **implies** ellenőrzése

a **multiplicitás** felső korlátjának vizsgálata

```
if |oktatók| = 0 then return error endif
if |hallgatók| = max then return error endif
if hallg in hallgatók then return error endif
hallgatók.Add(hallg);
```

mindkét **unique** feltételt ugyanaz az eldöntés (lineáris keresés) ellenőrzi

C# kód



```
public int Max { get; private set; }
private readonly List<Student> students = new (); // 0 .. max
private readonly List<Teacher> teachers = new (); // 0 .. 2

public Course(int max) { Max = max; }

public void AddStudent(Student student)
{
    if (teachers.Count == 0) throw new NoTeacherInThisCourse();
    if (students.Count == Max) throw new TooManyStudentsLeadThisCourse();
    if (students.Contains(student)) throw new StudentHasSignedUpThisCourse();
    students.Add(student);
}
```

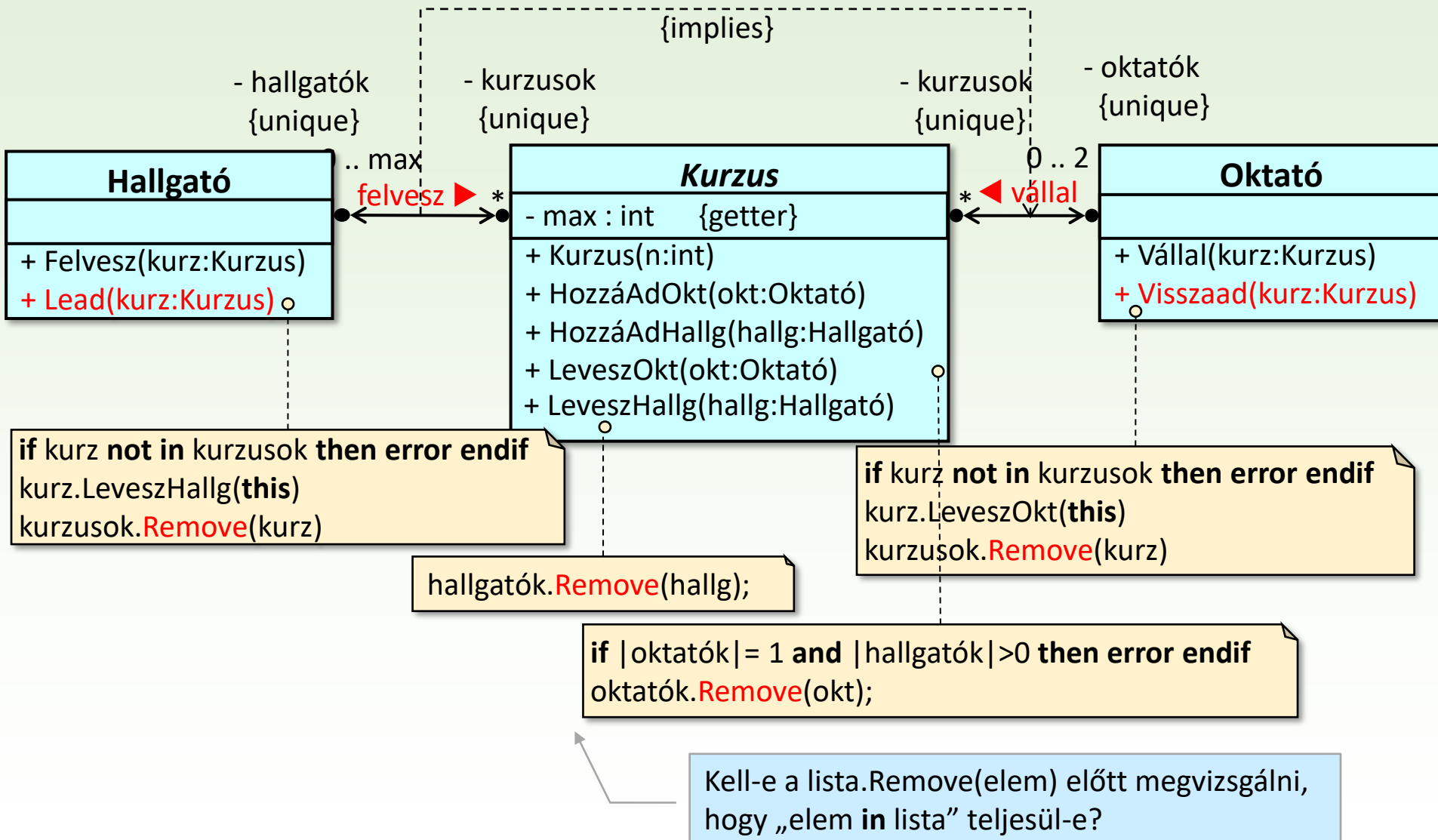
class Course

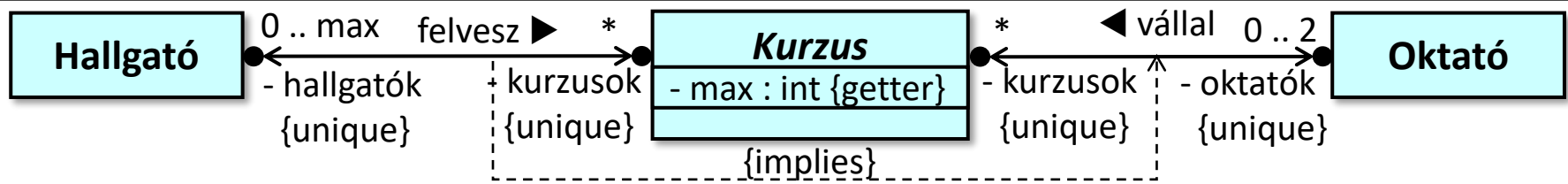
```
private readonly List<Course> courses = new ();

public void SignsUp(Course course)
{
    course.AddStudent(this);
    courses.Add(course);
}
```

class Student

Kapcsolatok bontása





```

private readonly List<Course> courses
    = new ();
public void SignsUp(Course course) {...}
public void SignsDown(Course course)
{
    if (!courses.Contains(course))
        throw new StudentHasNotSuchCourse();
    course.RemoveStudent(this);
    courses.Remove(course);
}

```

class Student

```

private readonly List<Course> courses
    = new ();
public void Undertakes(Course course){...}
public void Drops(Course course)
{
    if (!courses.Contains(course))
        throw new StudentHasNotSuchCourse();
    course.RemoveTeacher(this);
    courses.Remove(course);
}

```

class Teacher

```

private readonly List<Student> students = new (); // 0 .. max
private readonly List<Teacher> teachers = new (); // 0 .. 2

public void RemoveTeacher(Teacher t)
{
    if (teachers.Count == 1 && students.Count > 0) throw new TeacherCannotRemoved();
    teachers.Remove(t);
}

public void RemoveStudent(Student student)
{
    students.Remove(student);
}

```

class Course

Modell megvalósítása

3.rész

Kurzus-számonkérés
kapcsolatok

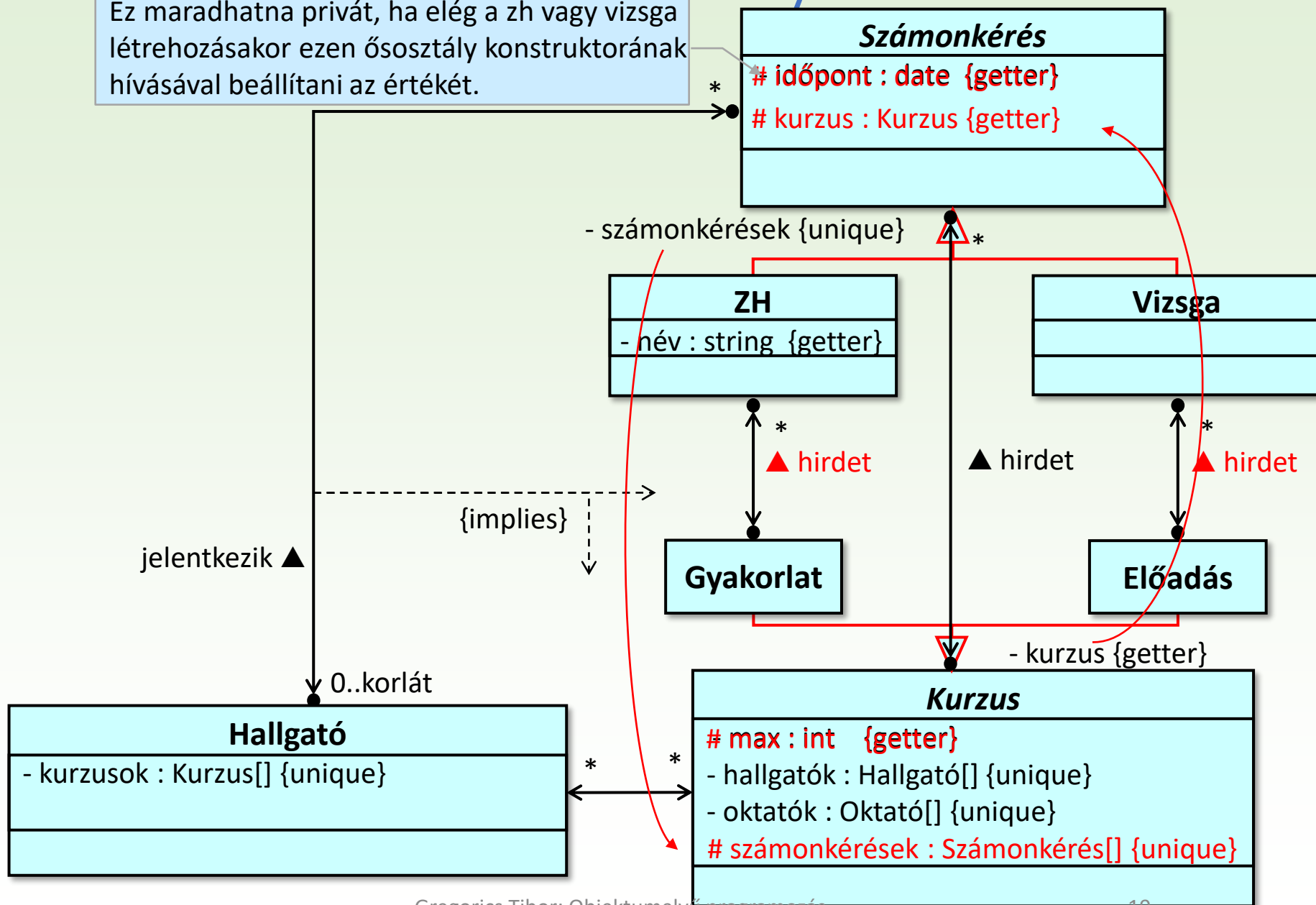
Gregorics Tibor

gt@inf.elte.hu

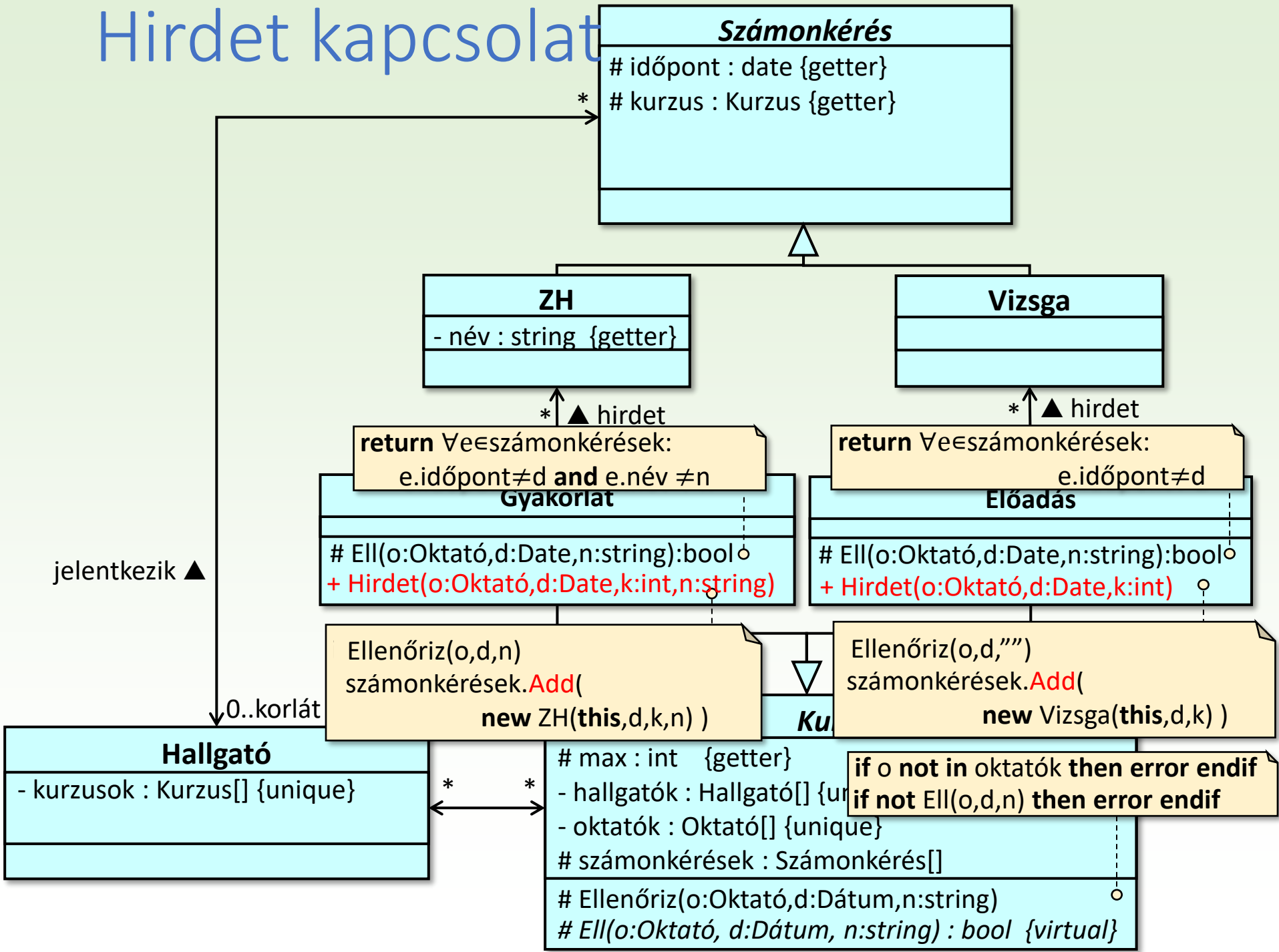
<http://people.inf.elte.hu/gt/oep>

Számonkérések osztályai

Ez maradhatna privát, ha elég a zh vagy vizsga létrehozásakor ezen őssztály konstruktorának hívásával beállítani az értékét.



Hirdet kapcsolat



C# kód

```
protected abstract bool Check(DateTime date, string name = "");  
protected bool CheckAnnounce(Teacher teacher, DateTime date, string name = "")  
{  
    if (!teachers.Contains(teacher)) throw new TeacherHasNoPermission();  
    if (!Check(date, name)) throw new Control.WrongControlParameters();  
    return true;  
}
```

class Course

```
public Exam Announce(Teacher teacher, DateTime date, int limit)  
{  
    CheckAnnounce(teacher, date);  
    Exam exam = new (this, date, limit);  
    controls.Add(exam);  
    return exam;  
}
```

erre a tesztkörnyezetnek (főprogram) lesz szüksége

```
protected override bool Check(DateTime date, string name = "")  
{  
    foreach (Exam exam in controls)  
    {  
        if (exam.date == date) return false;  
    }  
    return true;  
}
```

class Lecture : Course

C# kód

```
protected abstract bool Check(DateTime date, string name = "");  
protected bool CheckAnnounce(Teacher teacher, DateTime date, string name = "")  
{  
    if (!teachers.Contains(teacher)) throw new TeacherHasNoPermission();  
    if (!Check(date, name)) throw new Control.WrongControlParameters();  
    return true;  
}
```

class Course

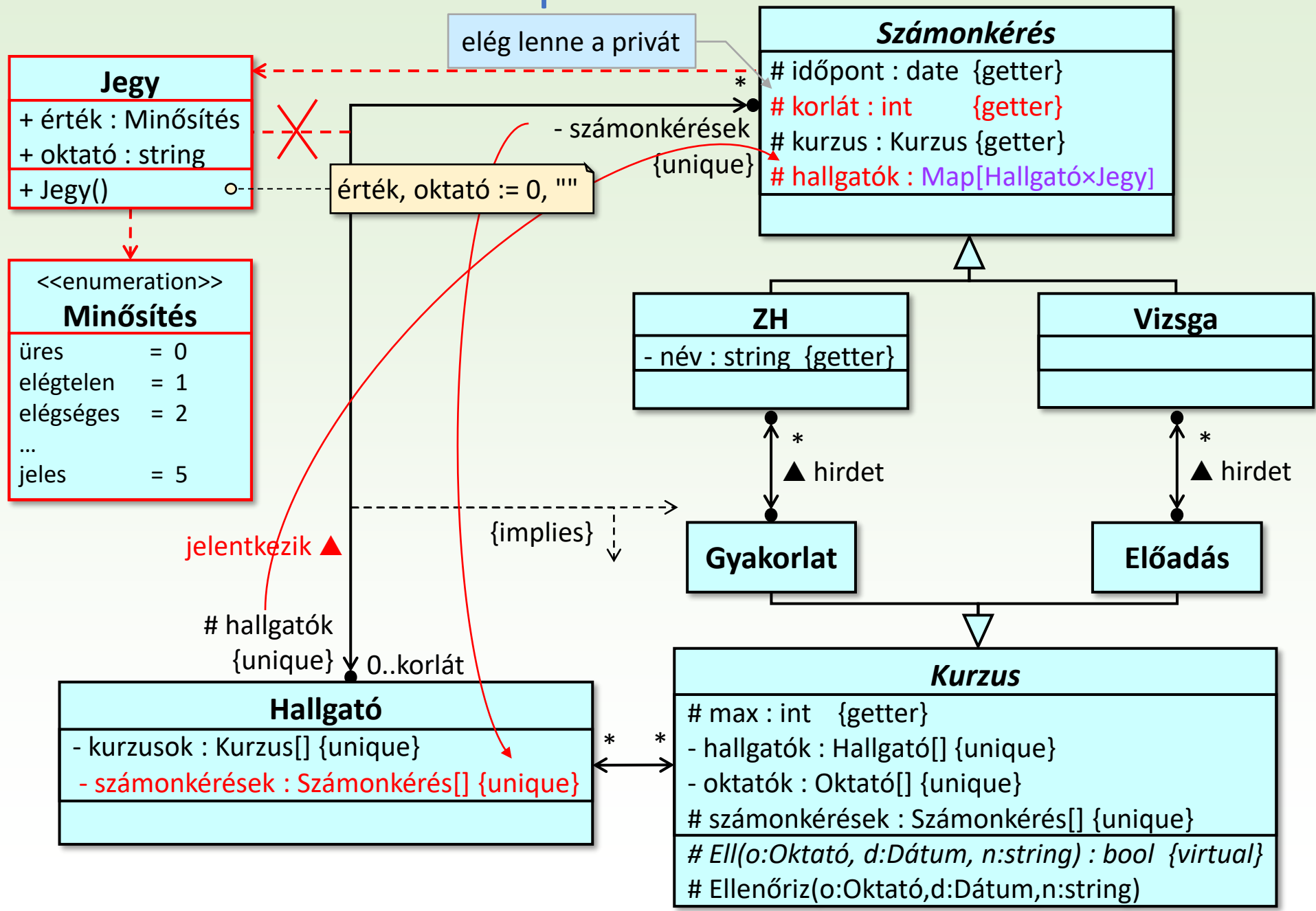
```
public Test Announce(Teacher teacher, DateTime date, int limit, string name)  
{  
    CheckAnnounce(teacher, date, name);  
    Test test = new (this, date, limit, name);  
    controls.Add(test);  
    return test;  
}
```

erre a tesztkörnyezetnek (főprogram) lesz szüksége

```
protected override bool Check(DateTime date, string name)  
{  
    foreach (Test test in controls)  
    {  
        if (test.name == name && test.date == date) return false;  
    }  
    return true;  
}
```

class Practice : Course

Jelentkezik kapcsolatot

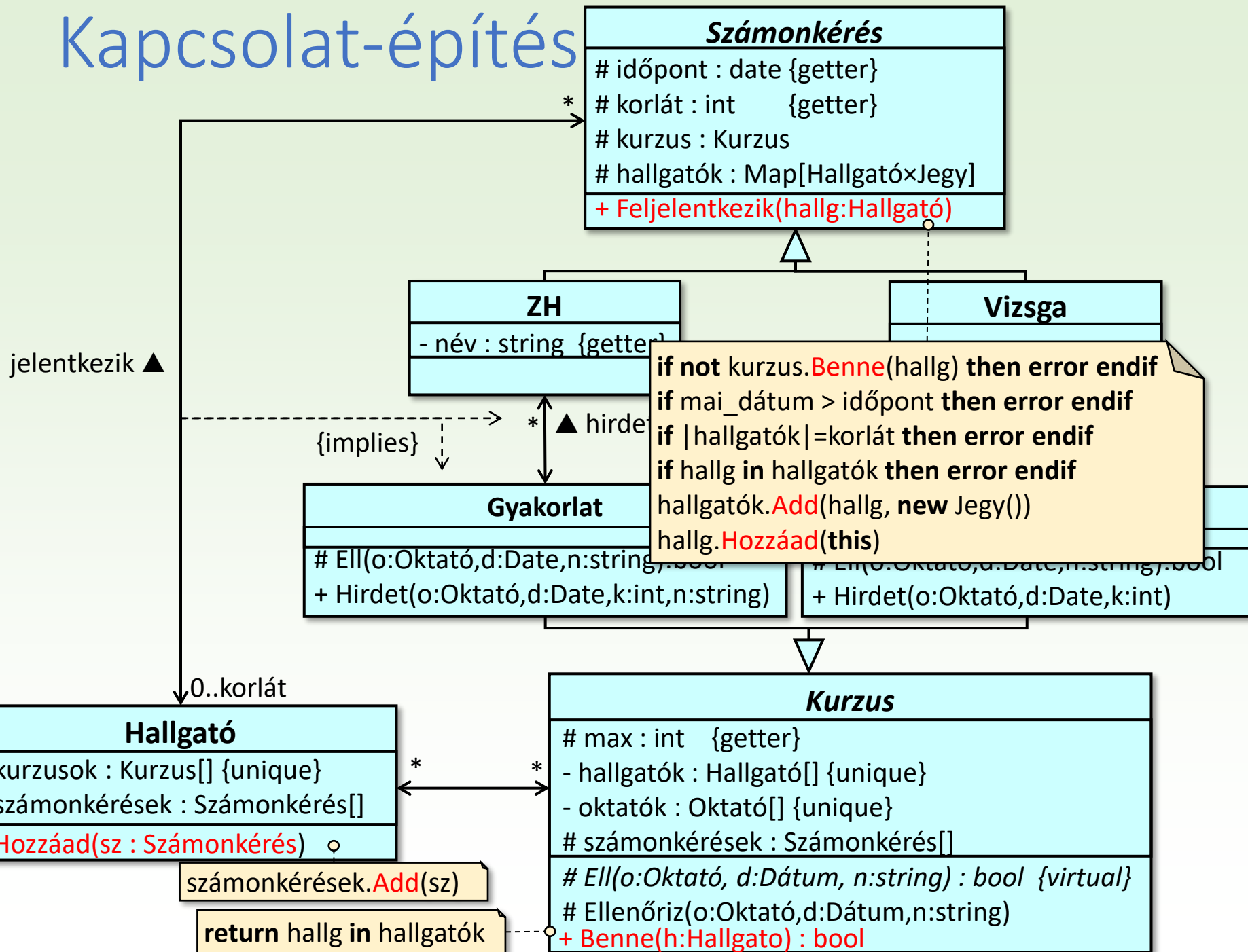


C# kód

```
enum Category { excellent = 5, good = 4, satisfactory = 3, poor = 2,
                insufficient = 1, no_grade = 0 }

class Grade
{
    public Category value;
    public string teacher;
    public Grade() { value = Category.no_grade; teacher = "" ; }
    public override string ToString()
    {
        return value.ToString() + " by " + teacher;
    }
}
```


Kapcsolat-építés



C# kód

```
public int Limit { get; protected set; }
protected readonly Dictionary<Student, Grade> students = new ();

public void Registrare(Student student)
{
    if (!course.In(student) ) throw new Course.StudentNotFoundInCourse();
    if (DateTime.Now > date ) throw new TimeLimit();
    if (students.Count == Limit) throw new NewStudentOverLimit();
    if (students.ContainsKey(student)) throw new StudentAlreadyRegistered();
    students.Add(student, new Grade());
    student.AddControl(this);
}
```

class Control

Dictionary metódusa

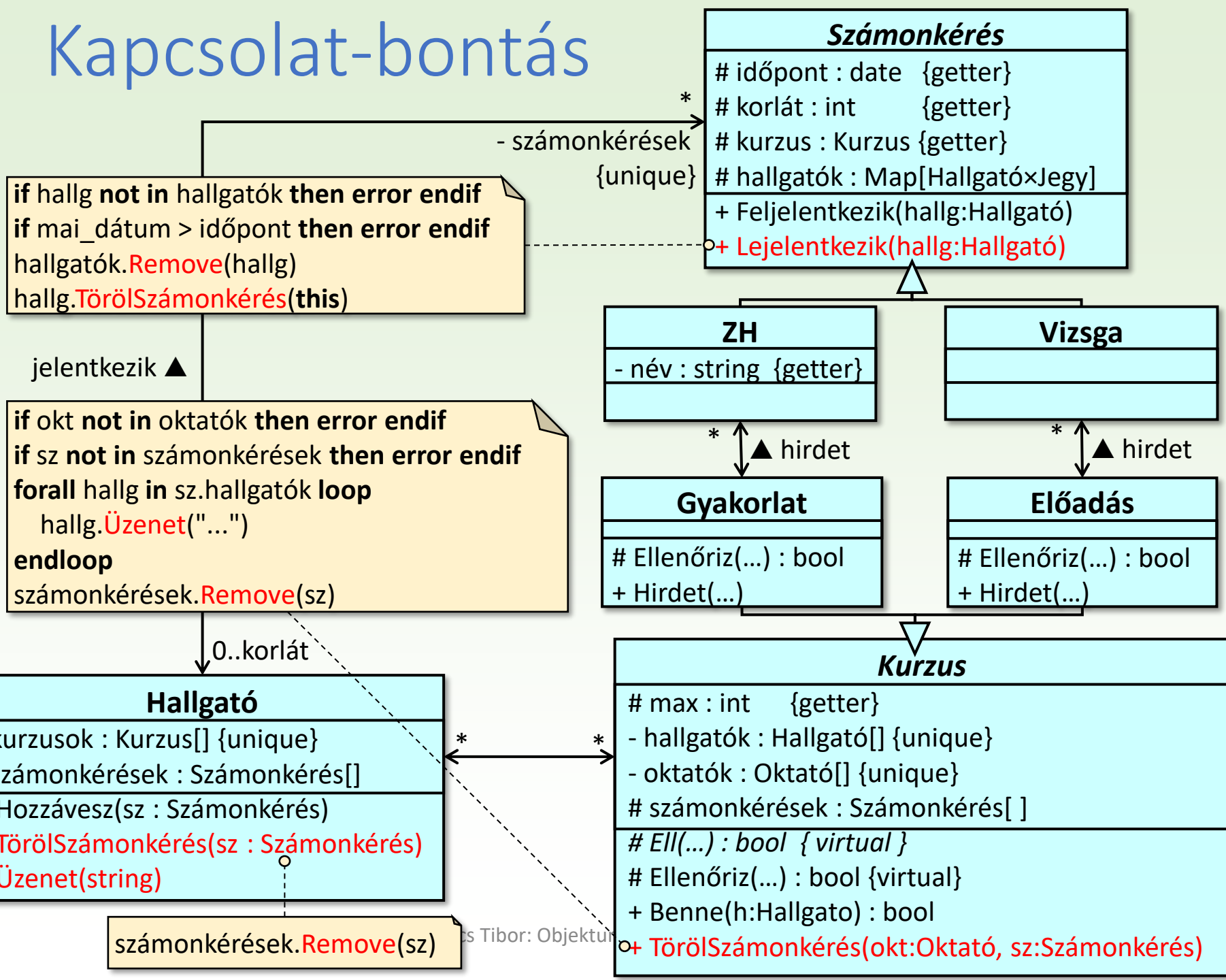
```
public void AddControl(Control ex)
{
    controls.Add(ex);
}
```

class Student : Person

```
public bool In(Student student)
{
    return students.Contains(student);
}
```

class Course

Kapcsolat-bontás



C# kód

```
public void DeleteControl(Teacher teacher, Control control)
{
    if (!teachers.Contains(teacher)) throw new TeacherHasNoPermission();
    if (!controls.Contains(control)) throw new ControlNotFoundInCourse();
    foreach (Student e in students) e.Delete(control);
    controls.Remove(control);
}
```

class Course

```
public void Unregisterate(Student student)
{
    if (!students.ContainsKey(student)) throw new StudentNotRegistered();
    if (DateTime.Now > date ) throw new TimeLimit();
    students.Remove(student);
    student.DeleteControl(this);
}
```

class Control

Dictionary metódusa

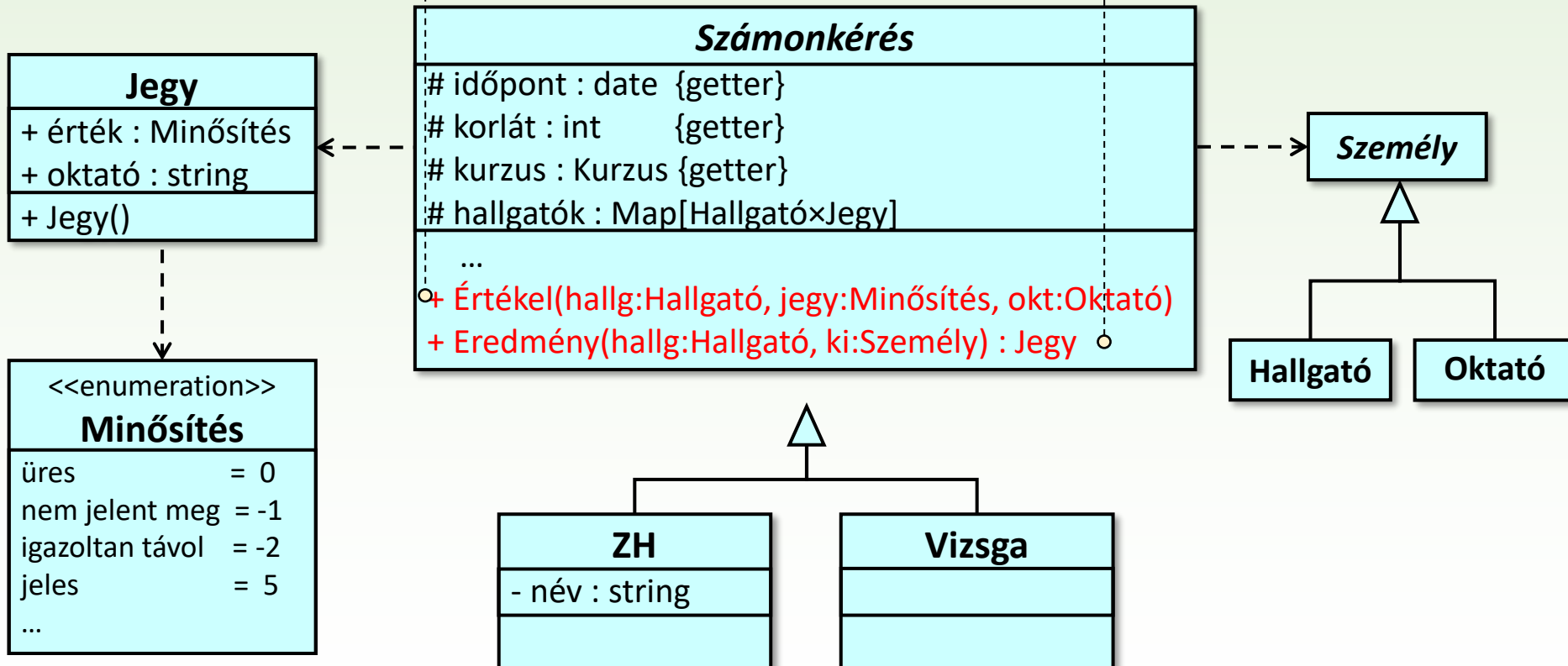
```
public void DeleteControl(Control ex)
{
    controls.Remove(ex);
}
```

class Student : Person

Értékelés és megtekintése

```
if okt not in kurzus.oktatók then error endif
if hallg not in hallgatók then error endif
hallgatók[hallg].érték := jegy
hallgatók[hallg].oktató := okt.név
```

```
if ki≠hallg or ki not in kurzus.oktatók then error endif
if hallg not in hallgatók then return null endif
return hallgatók[hallg]
```



C# kód

kurzus oktatói listájának getter-e

```
public void Evaluate(Student student, Category category, Teacher teacher)
{
    if (!course.Teachers.Contains(teacher)) throw new Course.TeacherFoundInCourse();
    if (!students.ContainsKey(student)) throw new StudentNotRegistered();
    students[student].value = category;
    students[student].teacher = teacher.Name;
}
```

```
public Grade Result(Student student, Person person)
{
    if ( person is Student s && !student.Equals(s) )
        throw new Course.StudentHasNoPermission();
    if ( person is Teacher t && !course.Teachers.Contains(t) )
        throw new Course.TeacherHasNoPermission();
    if (!students.ContainsKey(student)) return new Grade();
    return students[student];
}
```

kurzus oktatói listájának getter-e

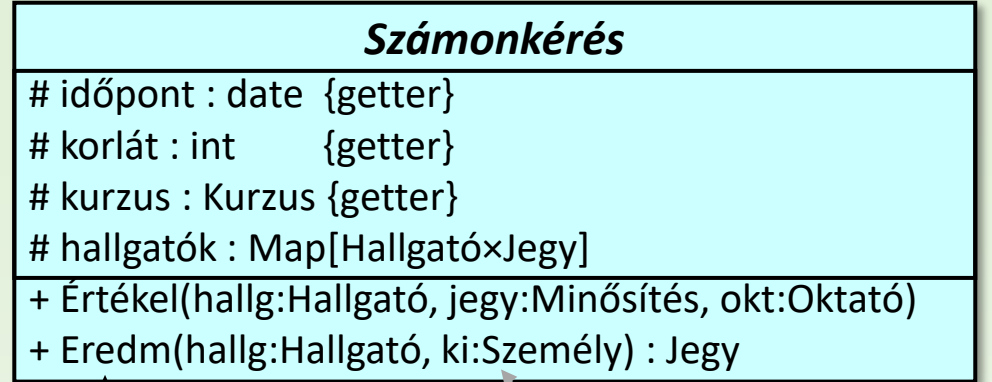
class Control

Alternatív nyelvi elemek:

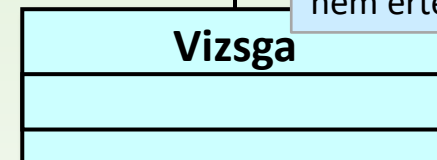
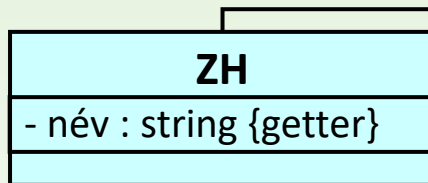
típusellenőrzés: `person.GetType().Equals(typeof(Student))`

konverzió: `t = (Teacher)person`

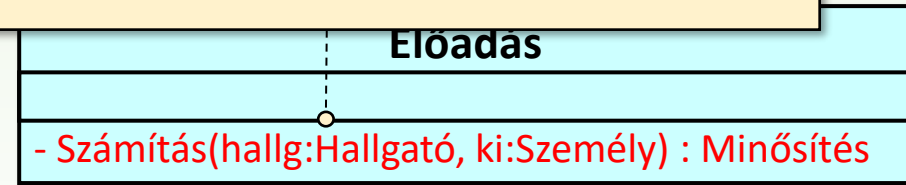
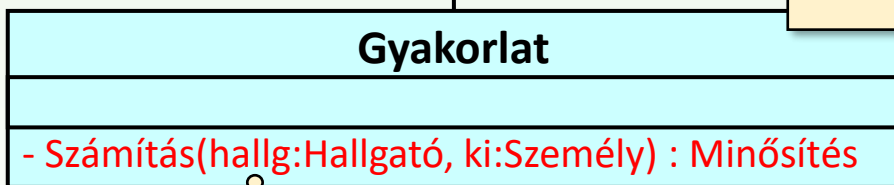
Jegy kiszámítása



0 értékű, ha a hallgatót nem értékelték

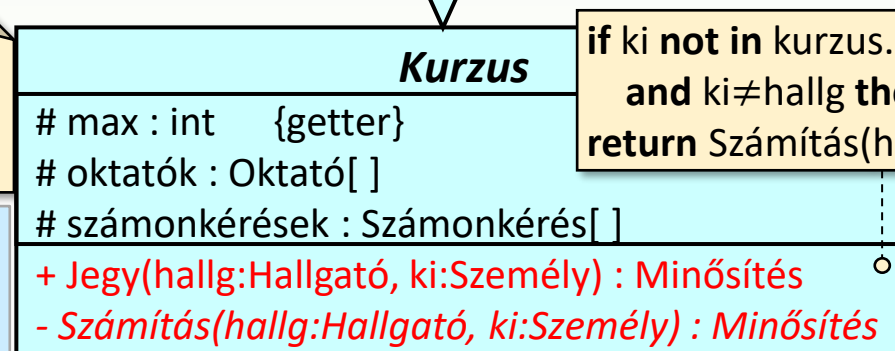


if |számonkérések|=0 then return 0 endif
return $\text{MAX}_{e \in \text{számonkérések}} e.\text{Eredm}(\text{hallg}, ki).\text{érték}$



$\text{össz, db} := \sum_{e \in \text{enor}} (e, 1)$
if db=0 then error endif
return $\text{össz}/\text{db}$

a hallgató zárthelyijeinek minősítéseit sorolja fel, de az azonos nevű zárthelyik közül csak a legjobbat



if ki not in kurzus.oktatók
and ki≠hallg then error endif
return Számítás(hallg, ki)

C# kód

```
public Category Grade(Student student, Person person)
{
    if (person is Teacher t && !Teachers.Contains(t) )
        throw new TeacherHasNoPermission();
    if (person is Student s && !student.Equals(s) )
        throw new StudentHasNoPermission();
    return ComputeGrade(student, person);
}

public abstract Category ComputeGrade(Student student, Person person);
```

class Course

C# kód

a legjobb vizsgajegyet keressük, de ehhez nem a maximum kiválasztás mintát, hanem az összegzés mintát használjuk

```
public override Category ComputeGrade(Student student, Person person)
{
    Category max = 0;
    foreach (Exam exam in controls)
    {
        int res = exam.Result(student, person).value;
        if (max < res) max = res;
    }
    return max;
}
```

```
class Lecture : Course
```

C# kód

Ötlet: a hallgató különböző nevű zh-kon elért 0-nál jobb jegyeit tároljuk a zh nevével azonosítva, de az azonos nevű zh-kra kapott jegyek közül mindig csak a legjobbat.

```
public override Category ComputeGrade(Student student, Person person)
{
    int sum = 0;
    Dictionary<string, int> results = new ();
    foreach (Test test in controls)
    {
        int res = Convert.ToInt32(test.Result(student, person).value);
        if (!results.ContainsKey(test.name)) {
            sum += res;
            results.Add(test.name, res);
        } else if (results[test.name] < res) {
            sum = sum - results[test.name] + res;
            results[test.name] = res;
        }
    }

    if (results.Count == 0) return Category.no_grade;
    else return (Category)(
        Convert.ToInt32(Math.Round(Convert.ToDouble(sum) / results.Count, 0)));
}
```

a minősítések számértékével kell dolgozni

Ötlet: ha a hallgató az adott nevű zh-ra már kapott, de rosszabb jegyet, akkor lecseréljük azt a jobbra a tárolóban is, és az összegben is

a visszaadott érték minősítés legyen

Class Practice : Course

Modell megvalósítása

4.rész

Modell tesztelése és
futtatási környezete

Gregorics Tibor

gt@inf.elte.hu

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

Egység tesztek tervezése

Az Eredmény() metódus vagy a korábban rögzített értékelés minősítését adja vissza a javító oktató nevével, de ha a hallgató nem vett részt a számonkérésen vagy még nem lett értékelve a munkája, akkor (0,"")-et.

❑ Ha egy metódus **adatokat kérdez le**, de nem módosít adatokat, akkor végezzünk fekete doboz tesztelést, hiszen ilyenkor a metódus által visszaadott értékeket kell csak ellenőrizni.

Jegy() metódusban egy összegzés található

❑ Ha azonban egy metódus tevékenysége **nevezetes algoritmus mintára épül**, akkor az algoritmus mintára jellemző szürke doboz teszteseteket is próbáljuk ki.

A Visszaad() metódus kivételt dob, ha olyan oktató hívja, aki egymaga vállalta el a kurzust, és már vannak hallgatói.

❑ A metódusokban végzett **hibaellenőrzés** kivétel-dobásokat general, és egy-egy tesztadattal ezek mindegyikét ki kell tudnunk váltani.

❑ A **kapcsolatokat** létrehozó vagy megszüntető metódusok helyes működését azokkal a metódusokkal vizsgálhatjuk, amelyek működése az ilyen kapcsolatokra épül, és egy nem létező kapcsolat esetén kivételt dob. E kivétel-dobás vagy annak elmaradása a kapcsolat hiányát vagy fennállását jelzi.

Az Értékel() metódussal olyan hallgatót is értékeljünk, aki nem jelentkezett a számonkérésre.

Teszt nevezetes algoritmus mintára

Maximum kiválasztás tesztesei:

- felsoroló hossza szerint
 - egy / több vizsgajegy
- maximum helye a felsorolásban
 - elején, végén, közben
- több egyforma maximum

Összegzés tesztesei:

- felsoroló hossza szerint
 - nulla / egy / több vizsgajegy
- összegzés művelete szerint
 - neutrális elem

```
public override Category Grade(Student student, Person person)
{
    Category max = 0;
    foreach (Exam exam in controls)
    {
        int res = exam.Result(student, person).value;
        if (max < res) max = res;
    }
    return max;
}
```

```
class Lecture : Course
```

Teszt nevezetes algoritmus mintára

Speciális tesztesetek:

- csupa eltérő nevű zh
- csak azonos nevű zh-k
- vegyes

Összegzés tesztesetei:

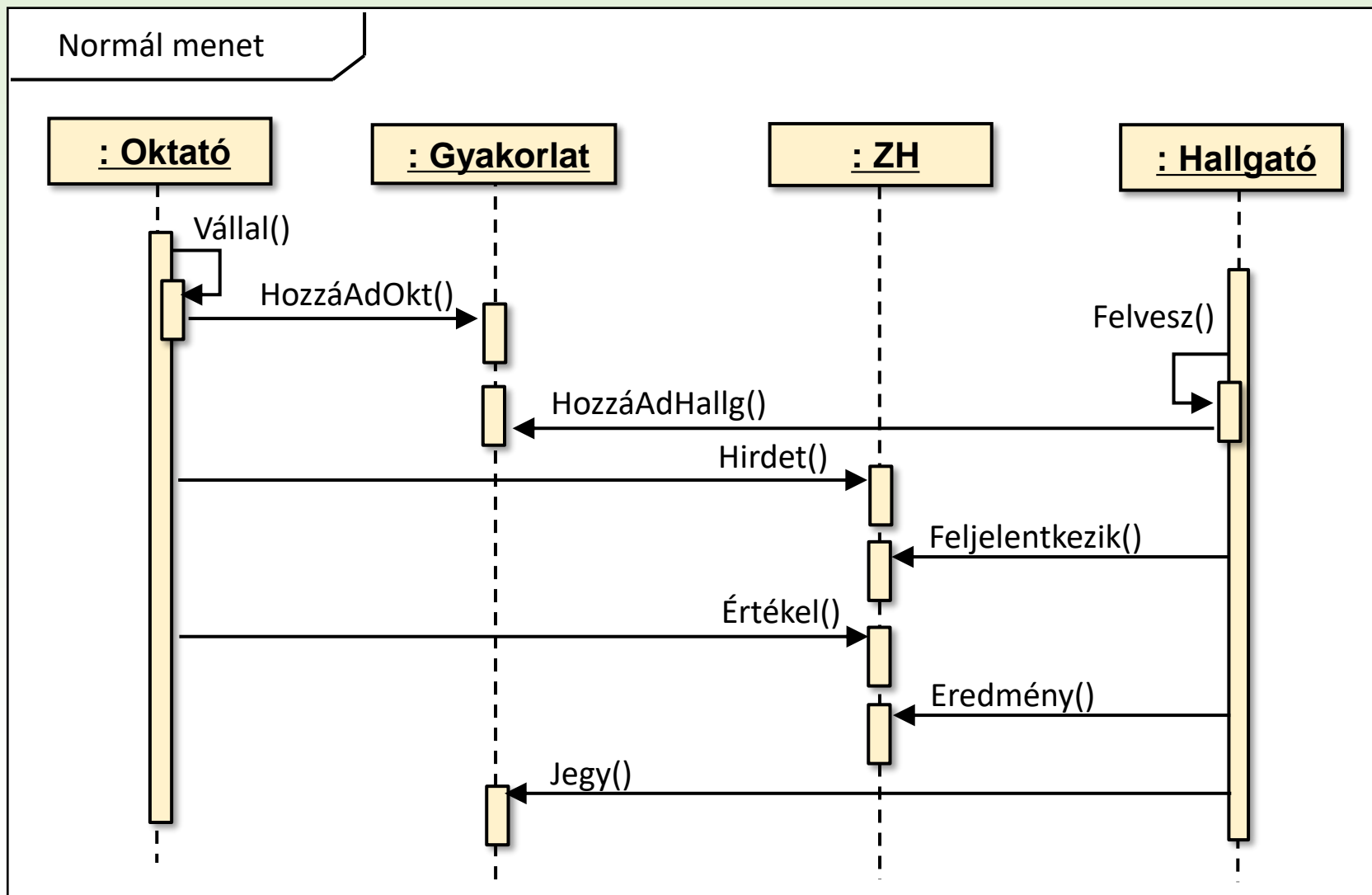
- felsoroló hossza szerint
 - nulla / egy / több zh
- összegzés művelete szerint
 - neutrális elem

```
public override Category ComputeGrade(Student student)
{
    Dictionary<string, int> results = new ();
    int sum = 0;
    foreach (Test test in controls)
    {
        int res = Convert.ToInt32(test.Result(student, person).value);
        if (!results.ContainsKey(test.name)) {
            sum += res;
            results.Add(test.name, res);}
        else if (results[test.name] < res) {
            sum = sum - results[test.name] + res;
            results[test.name] = res;
        }
    }

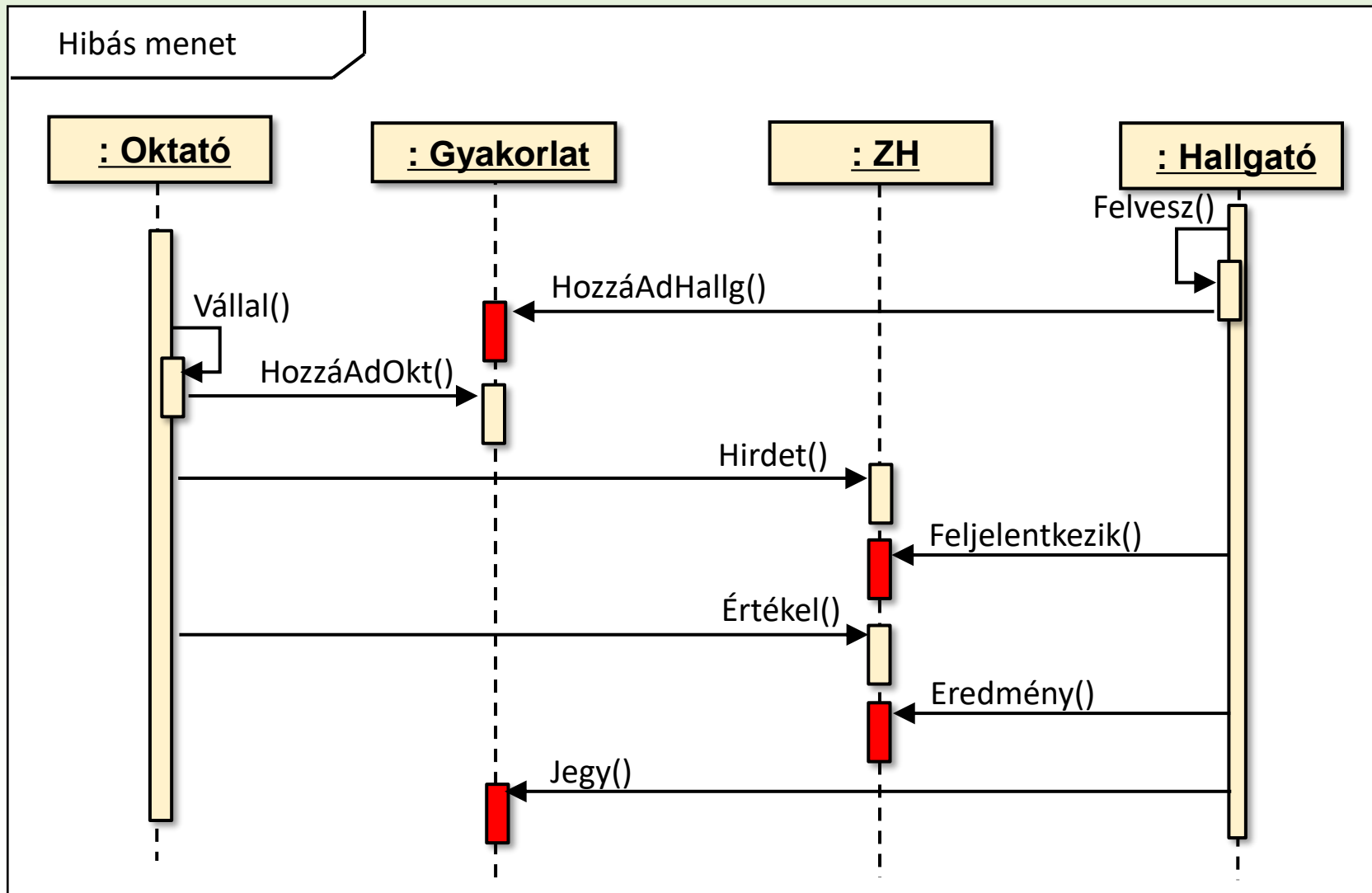
    if (results.Count == 0) return Category.no_grade;
    else return (Category)(
        Convert.ToInt32(Math.Round(Convert.ToDouble(sum) / results.Count, 0)));
}
```

```
class Practice : Course
```

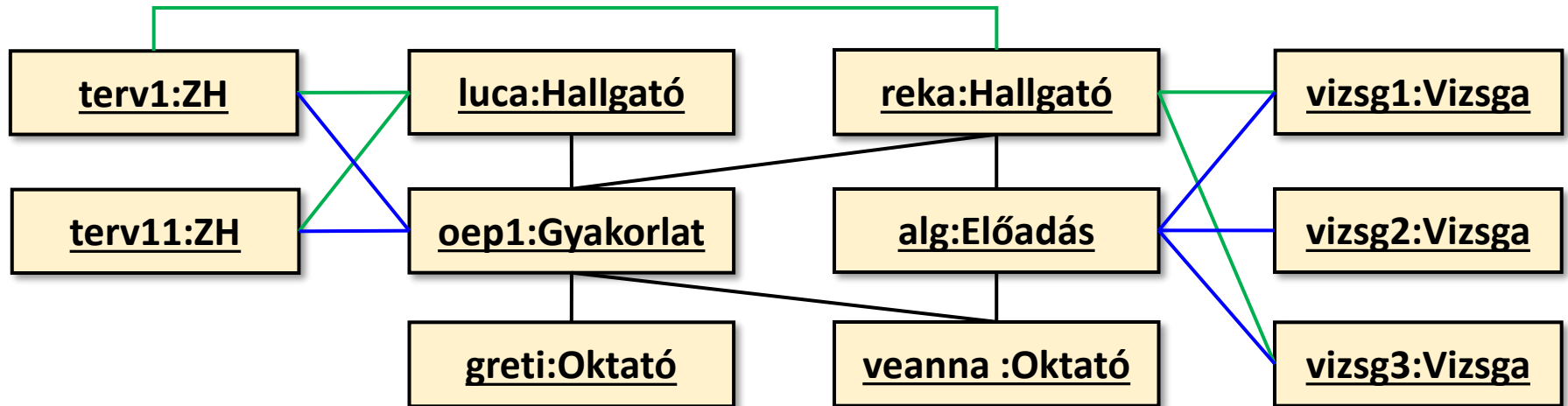
Integrációs tesztek tervezése



Integrációs tesztek tervezése



Felpopulálás



```
Course oep1 = new Practice("OEP1", 20);
Course alg  = new Lecture("ALG", 350);
Student luca = new ("Luca", "AAAAAA");
Student reka = new ("Reka", "BBBBBB");
Teacher veanna = new ("Anna", "DDDDDD");
Teacher greti = new ("Tibi", "CCCCC");
```

```
greti.Undertakes(oep1);
veanna.Undertakes(oep1);
veanna.Undertakes(alg);
```

```
luca.SignsUp(oep1); luca.SignsUp(alg);
reka.SignsUp(oep1); reka.SignsUp(alg);
```

```
Test terv1 = oep1.Announce(greti, DateTime.Parse("2023.03.30"), "1.terv");
Test terv1jav = oep1.Announce(greti, DateTime.Parse("2023.05.18"), "1.terv");
Exam exam1 = alg.Announce(veanna, DateTime.Parse("2023.05.15"), 25);
Exam exam2 = alg.Announce(veanna, DateTime.Parse("2023.05.15"), 25);
Exam exam3 = alg.Announce(veanna, DateTime.Parse("2023.05.15"), 25);
```

```
terv1.Registrate(luca);
terv1jav.Registrate(luca);
terv1.Registrate(reka);
exam1.Registrate(reka);
exam3.Registrate(reka);
```

Felpopulálás szöveges fájlból

feltételezzük, hogy a fájl formája helyes

```
PRA oep1 20
LEC alg 350
TEA veanna AAAAAA aaa
TEA gretiBBBBBB bbb
STU luca CCCCCC xxx
STU reka DDDDDD yyy
UNTA veanna alg
UNTA greti oep1
SIUP luca oep1
SIUP luca alg
SIDO luca alg
SIUP reka oep1
SIUP reka alg
ANNO veanna oep1 terv1 2023.03.30 20 1.terv
ANNO veanna oep1 terv11 2023.05.18 20 1.terv
ANNO veanna alg exam1 2023.05.15 25
ANNO veanna alg exam2 2023.05.22 25
ANNO veanna alg exam3 2023.05.29 25
DELE veanna alg exam2
...
```

kurzusok

oktatók és hallgatók

kurzusok elvállalása, visszaadása,
felvétele és leadása

számonkérések
hirdetése, törlése

```
...
REGI luca terv1
REGI luca terv11
REGI reka terv1
REGI reka exam1
REGI reka exam3
```

```
EVAL veanna terv1 luca 4
EVAL veanna terv11 luca 5
EVAL veanna exam1 reka 1
EVAL veanna exam1 reka 4
RESU luca luca terv1
QUAL luca luca oep1
...
```

számonkérések
értékelése

eredmények lekérése

PRA ~ gyakorlat

LEC ~ előadás

TEA ~ oktató

STU ~ hallgató

UNTA ~ elvállalás

DROP ~ visszaadás

SIUP ~ felvétel

SIDO ~ leadás

ANNO ~ hirdetés

DELE ~ törlés

REGI ~ jelentkezés

UNRE ~ lejelentkezés

EVAL ~ értékelés

RESU ~ eredmény

QUAL ~ jegy

Előkészítés

```
static Dictionary<string, Person> persons = new ();
static Dictionary<string, Course> courses = new ();
static Dictionary<string, Control> controls = new ();

static Person Person(string name)
{
    if (!persons.ContainsKey(name)) throw new FileContentError();
    return persons[name];
}
static Course Course(string name)
{
    if (!courses.ContainsKey(name)) throw new FileContentError();
    return courses[name];
}
static Control Control(string name) )
{
    if (!controls.ContainsKey(name)) throw new FileContentError();
    return controls[name];
}
```

nyilvántartjuk a létrehozott objektumokat,
hogy a nevük alapján kereshessük őket

Szöveges állomány feldolgozása

```
static void Main()
{
    Person adm = new ("Admin", "000000");
    Centre centre = new (adm);

    TextFileReader reader = new ("input.txt");
    char[] separators = new char[] { ' ', '\t' };

    while (reader.ReadLine(out string line))
    {
        string[] tokens = line.Split(separators, StringSplitOptions.RemoveEmptyEntries);
        switch (tokens[0])
        {
            case "PRA": courses.Add(tokens[1],
                                    new Practice(tokens[1], Convert.ToInt32(tokens[2]))); break;
            case "LEC": courses.Add(tokens[1],
                                    new Lecture(tokens[1], Convert.ToInt32(tokens[2]))); break;
            case "TEA": Teacher t = new (tokens[1], tokens[2]);
                        centre.ModifyPassword(t, centre.Registrate(t, adm), tokens[3]);
                        t.ModifyPassword(tokens[2], tokens[3]);
                        persons.Add(tokens[1], t);
                        break;
            case "STU": Student s = new (tokens[1], tokens[2]);
                        centre.ModifyPassword(s, centre.Registrate(s, adm), tokens[3]);
                        s.ModifyPassword(tokens[2], tokens[3]);
                        persons.Add(tokens[1], s);
                        break;
            ...
        }
    }
}
```

amíg az utolsó sort be nem olvastuk

Tevékenységek

Ősosztály típusú hivatkozás alosztály típusúra kasztolása, ha tudjuk, hogy ez az alosztály objektumának a hivatkozása.

```
...
case "UNTA": ((Teacher)persons[tokens[1]]).Undertakes(Course(tokens[2])); break;
case "GIUP": ((Teacher)persons[tokens[1]]).Drops(Course(tokens[2])); break;
case "SIUP": ((Student)persons[tokens[1]]).Signalls(Course(tokens[2])); break;
case "SIDO": ((Student)persons[tokens[1]]).Signalls(Course(tokens[2])); break;
case "ANNO": Course course = Course(tokens[2]);
               Control ex = null;
               if ( course is Practice p ) ex=p.Announce((Teacher)persons[tokens[1]],
                                                           DateTime.Parse(tokens[4]), int.Parse(tokens[5]), tokens[6]);
               else if ( course is Lecture l ) ex=l.Announce((Teacher)persons[tokens[1]],
                                                             DateTime.Parse(tokens[4]), int.Parse(tokens[5]));
               controls.Add(tokens[3], ex);
               break;
case "DELE": Course(tokens[2]).DeleteControl((Teacher)persons[tokens[1]],
                                              Control(tokens[3])); break;
case "REGI": Control(tokens[2]).Registrate((Student)persons[tokens[1]]); break;
case "UNRE": Control(tokens[2]).Unregistrate((Student)persons[tokens[1]]); break;
case "EVAL": Control(tokens[2]).Evaluate((Student)persons[tokens[3]],
                                          (Category)int.Parse(tokens[4]), (Teacher)persons[tokens[1]]); break;
case "RESU": Console.WriteLine($"{Control(tokens[3]).Result((Student)Person(tokens[2]),
                                                             persons[tokens[1]])}"); break;
case "QUAL": Console.WriteLine($"{Course(tokens[3]).Grade((Student)Person(tokens[2]),
                                                           persons[tokens[1]])}"); break;
```

Annak vizsgálata, hogy egy ősosztály típusú változó az adott alosztály objektumára hivatkozik-e. Ha igen, akkor a hivatkozás egy alosztály típusú (p) változó értéke lesz.

idő előállítása sztringből

Hitelesítés

```
if sz.azon not in tábla or not jelszó=tábla[sz.azon] then error endif
```

nyilvántartjuk a személyek
azonosítóját és jelszavát

ki lehetne még egészíteni a
modellt adminisztrátorokkal

```
if not ( adm.azon in admin and  
         jelszó = admin[adm.azon] )  
then error endif  
if sz.azon in tábla then error endif  
új := generált jelszó  
tábla.Add(sz.azon, új)  
return új
```

Számonkérés

Kurzus

Központ

```
- tábla : MAP[string, string]  
- admin : MAP[string, string] { |admin|>0 }  
+ Hitelesít(sz:Személy, jelszó:String)  
+ Regisztrál(sz:Személy , adm:Személy, jelszó:string) : string  
+ JelszótMódosít(sz:Személy, új:string)
```

- központ

- központ

ezt minden olyan publikus metódus meghívja,
amelyik egy paraméterként kapott személy
nevében adatot módosít vagy lekérdez

```
Hitelesít(sz)  
tábla[sz.azon] := új
```