Kapcsolatok modellezése és megvalósítása

Gregorics Tibor

gt@inf.elte.hu

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

1.rész
Egy feladat elemzése

Gregorics Tibor

gt@inf.elte.hu

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

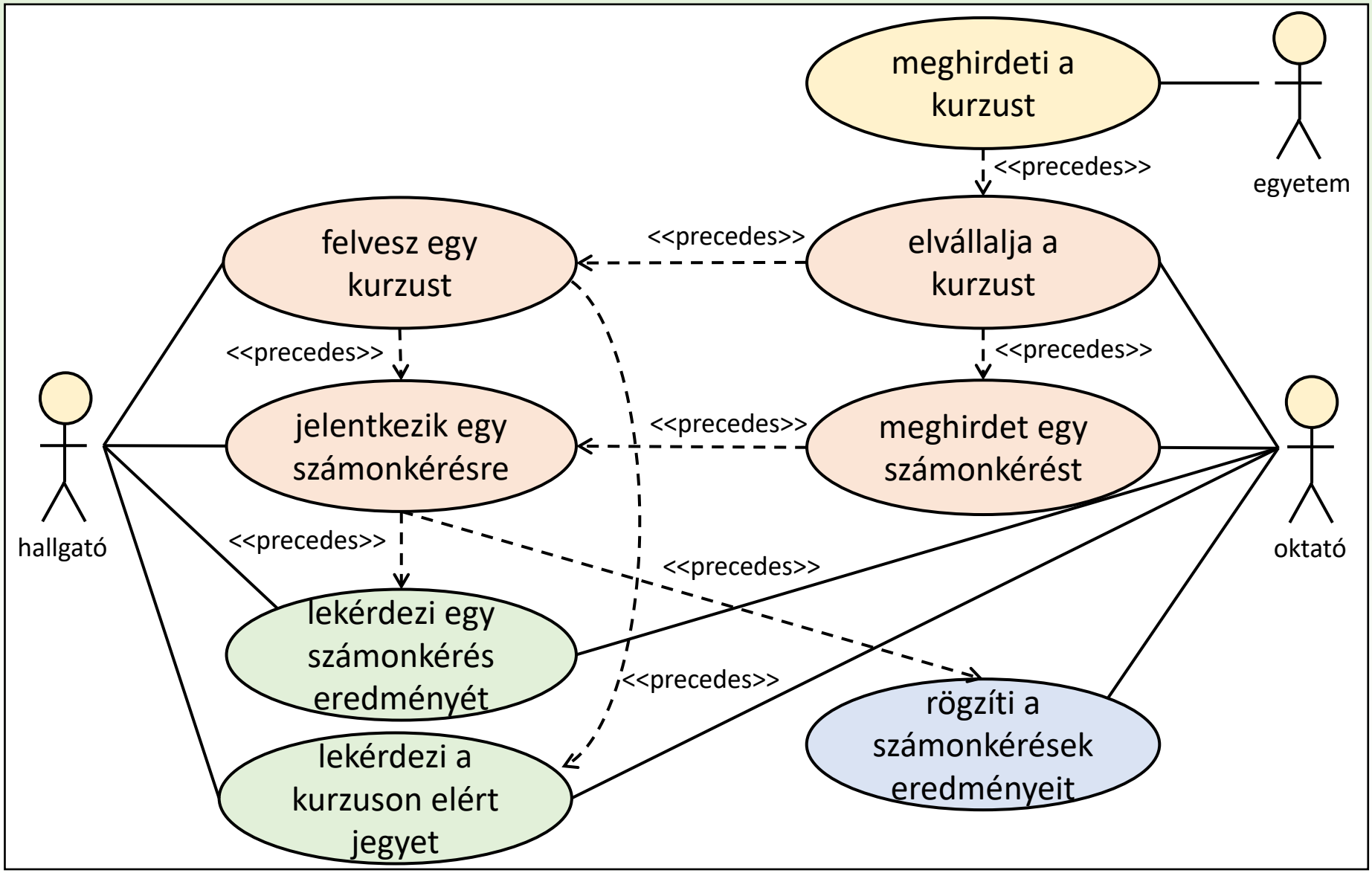
Feladat: Kurzusok

Egy egyetem kurzusokat hirdet meg, amelyek megtartására oktatók (legfeljebb ketten) vállalkozhatnak. A hallgatók a már oktatókkal rendelkező kurzusokra jelentkezhetnek (ha van még azon szabad hely).

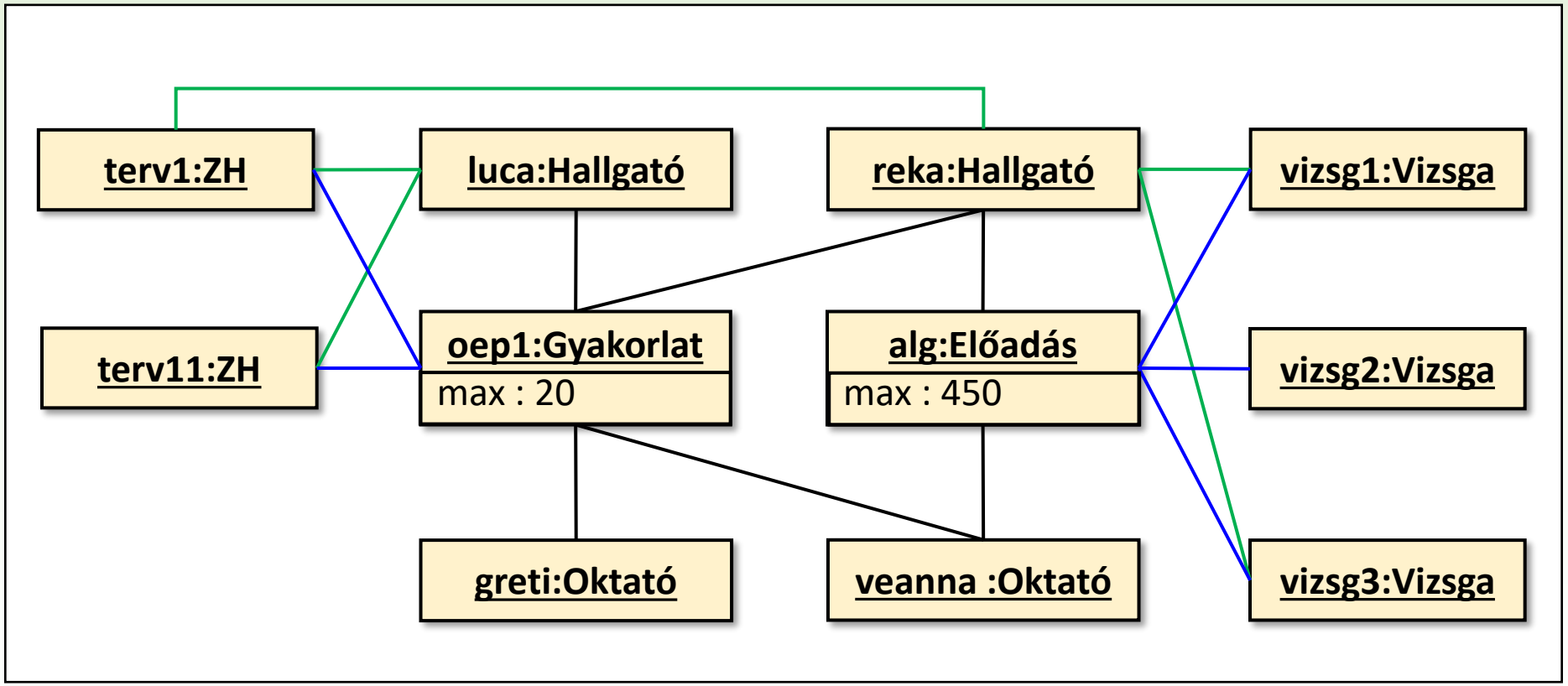
Egy kurzushoz annak jellegétől (előadás vagy gyakorlat) függően vizsgákat vagy zárthelyiket tűzhetnek ki a kurzus oktatói. A vizsgákat a dátumuk, a zárthelyiket a dátum mellett a nevük is azonosítja (a későbbi dátumúak a javító/pót zárthelyik). Az oktatók értékelik a számonkéréseket, és ezek eredménye alapján a hallgató jegyet kap a kurzusra.

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), de egy gyakorlat bármelyik zárthelyijére jelentkezhet (a későbbi dátumú azonos nevű zh-k a javító/pót zh-k). A hallgató megtekintheti a számonkérései eredményét, és lekérdezheti a kurzuson elért jegyét. Előadásnál a vizsgajegy a legutolsó vizsgán elért eredmény; a gyakorlati jegy a zárthelyi-eredmények átlaga, de úgy, hogy az ugyanolyan nevű zárthelyik közül mindig csak a legjobbik számít ebbe bele. A kurzus oktatói is láthatják a kurzus hallgatóinak eredményeit és jegyét.

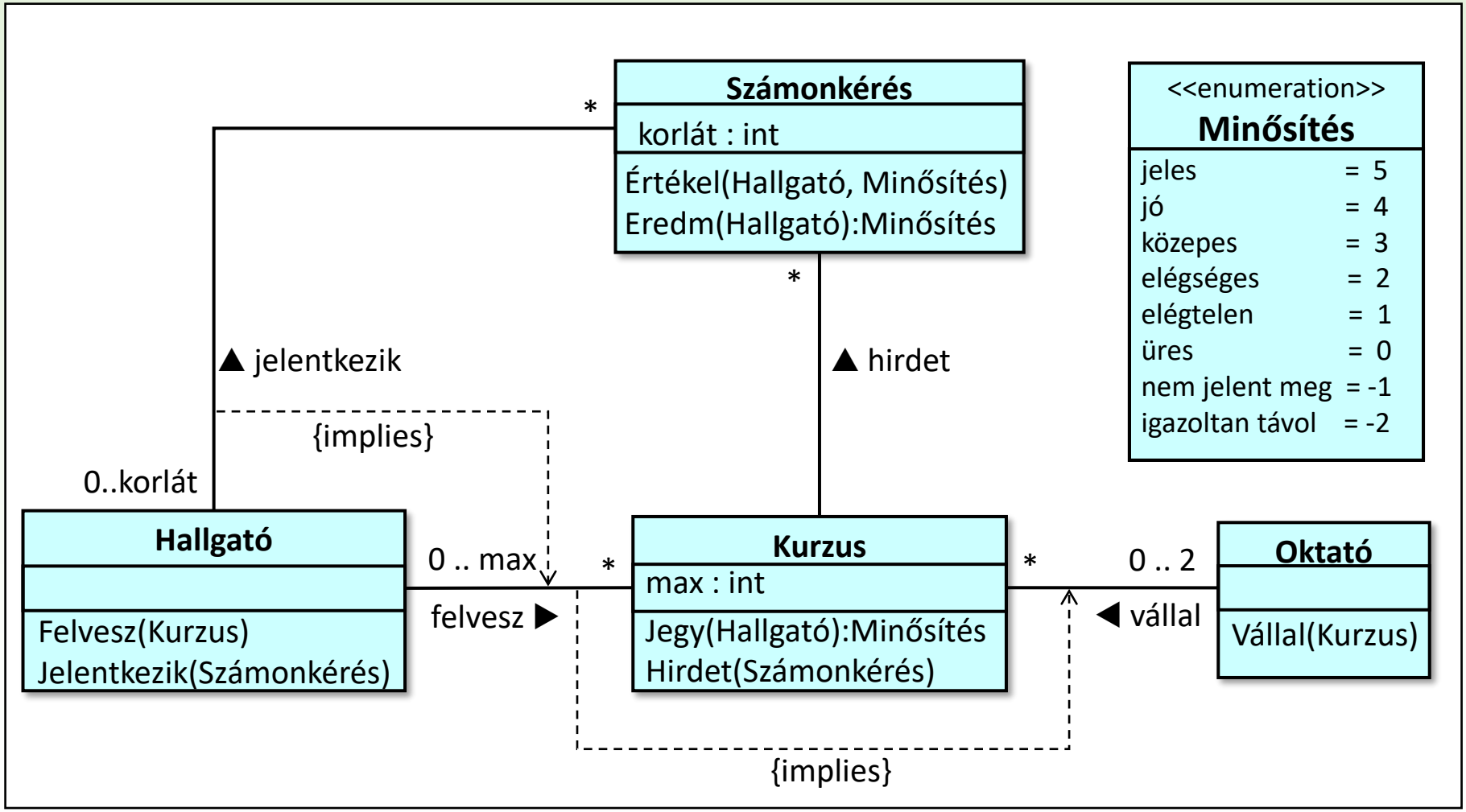
Felhasználói esetek



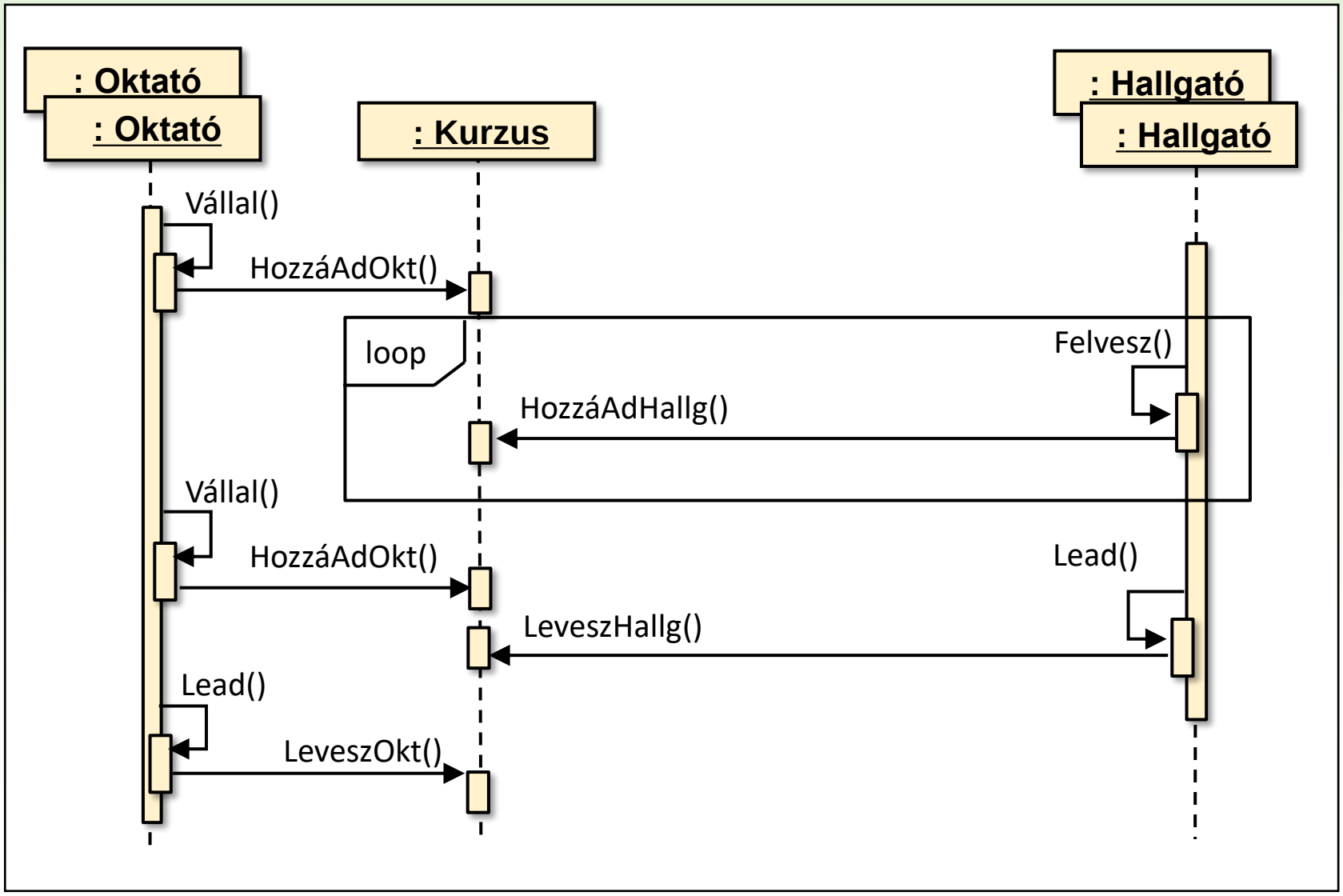
Objektumok és kapcsolataik



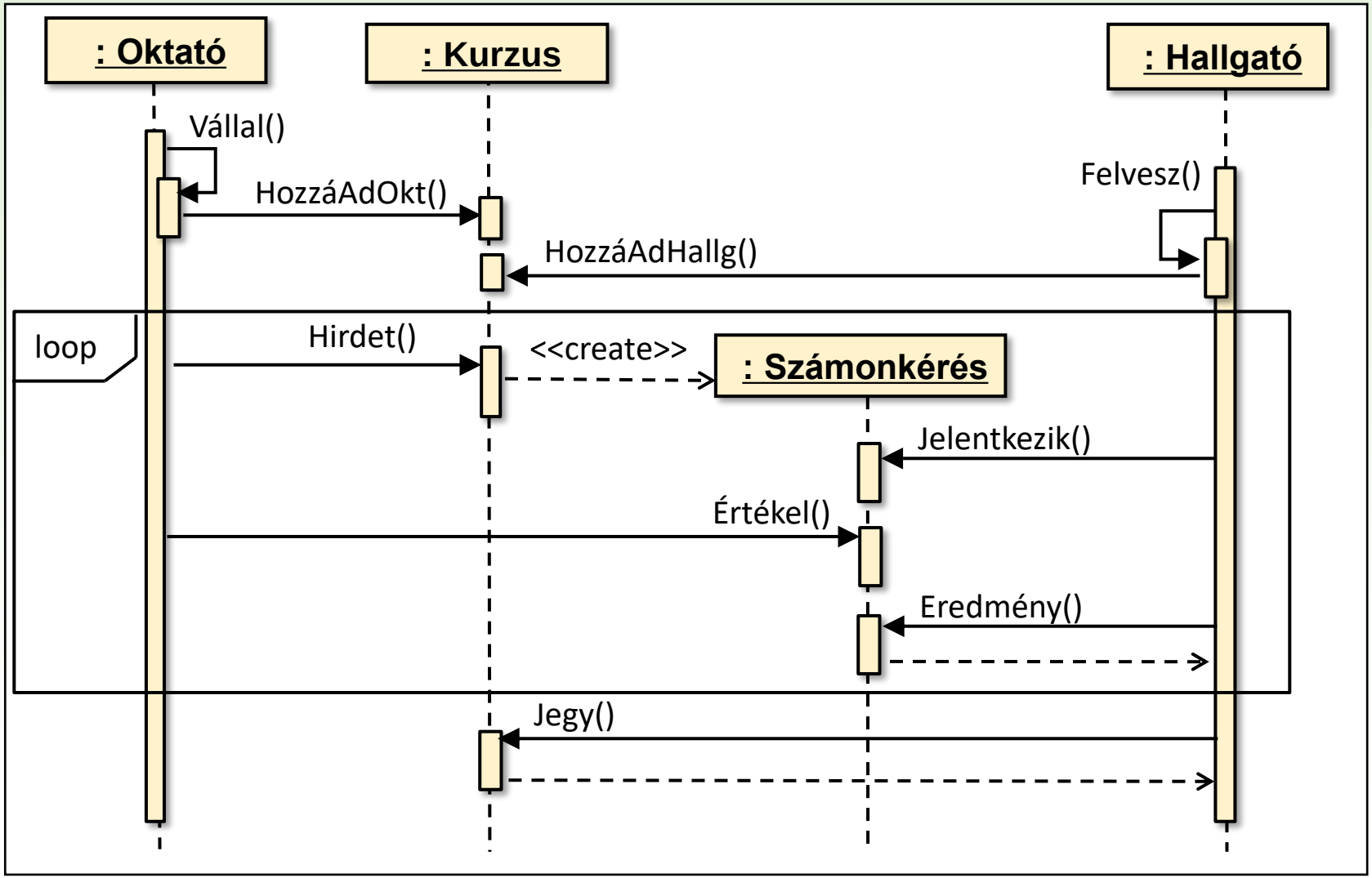
Osztály diagram



Csatlakozás egy kurzushoz



Számonkérések kezelése



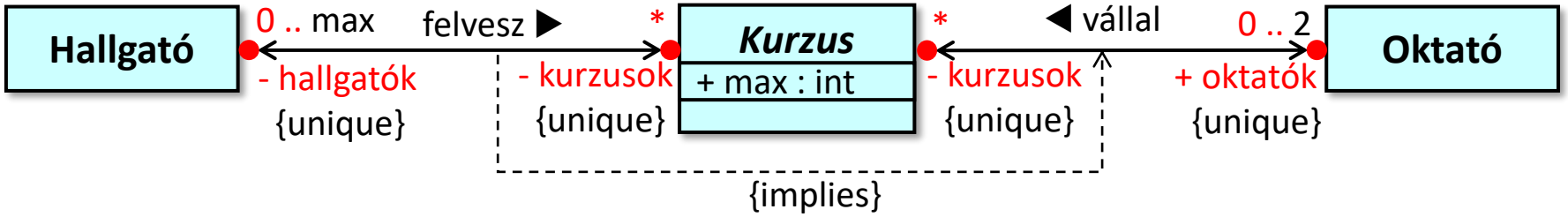
2.rész
Hallgató-kurzus-oktató kapcsolatok

Gregorics Tibor

gt@inf.elte.hu

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

Hallgatók, oktatók, kurzusok osztályai



A Max publikusan olvasható, de csak a Course metódusai módosíthatják. Ha a módosítást csak a konstruktornak engedjük meg, akkor `public int Max { get;}`, vagy a `public readonly int Max`.

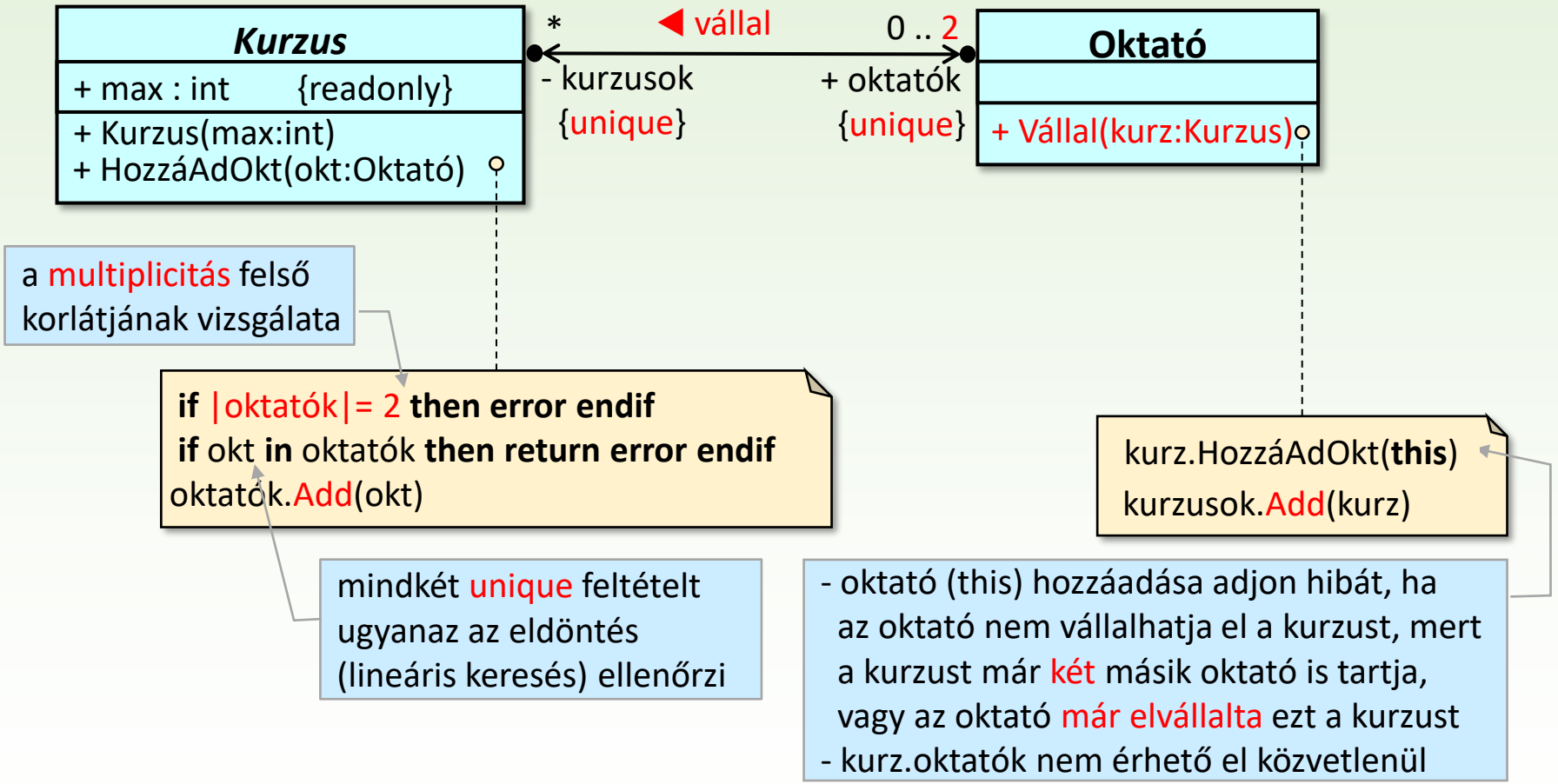
```
class Course
{
   public int Max { get; private set; }
   private readonly List<Student> students = new ();  // 0 .. max
   private readonly List<Teacher> teachers = new ();  // 0 .. 2
   public List<Teacher> Teachers { get { return teachers; } }
   public Course(int max) { Max = max; }
   ...
}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

ha publikussá tennénk, az nem lenne elég, mert hivatkozás típusú változónál a *readonly* csak a hivatkozásra vonatkozik, a hivatkozott objektum adataira (azaz a lista elemeire) nem.

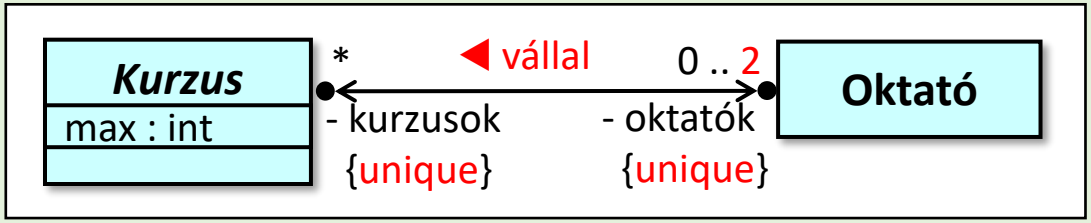
```
class Student
{
   private readonly List<Course> courses = new ();
   ...
}
```

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

Oktató-kurzus kapcsolatok építése



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);
}
```

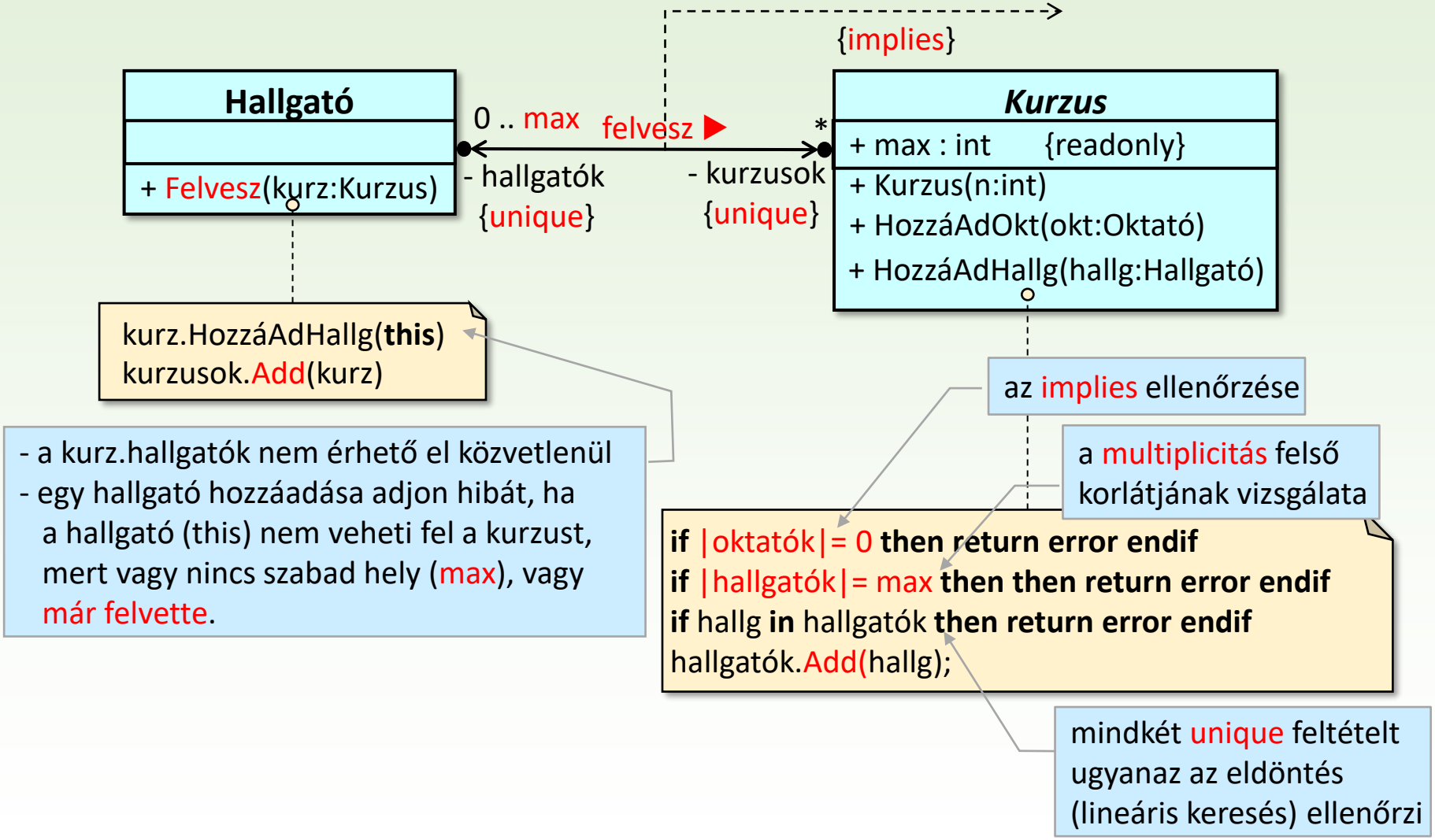
okt **in** oktatók

class Course

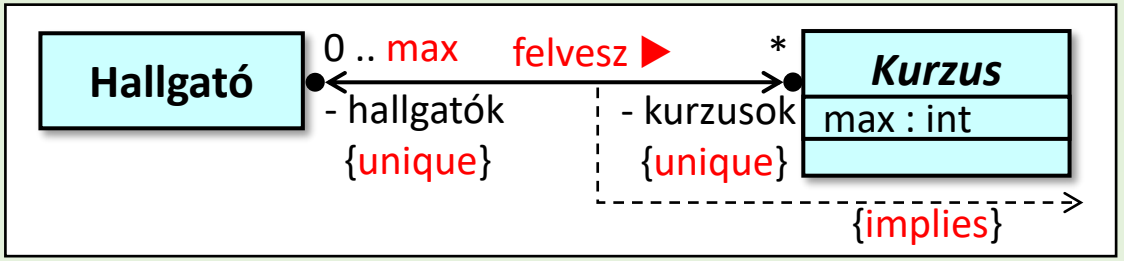
```
private readonly List<Course> courses = new ();

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

class Teacher

Hallgató-kurzus kapcsolatok építése



C# kód



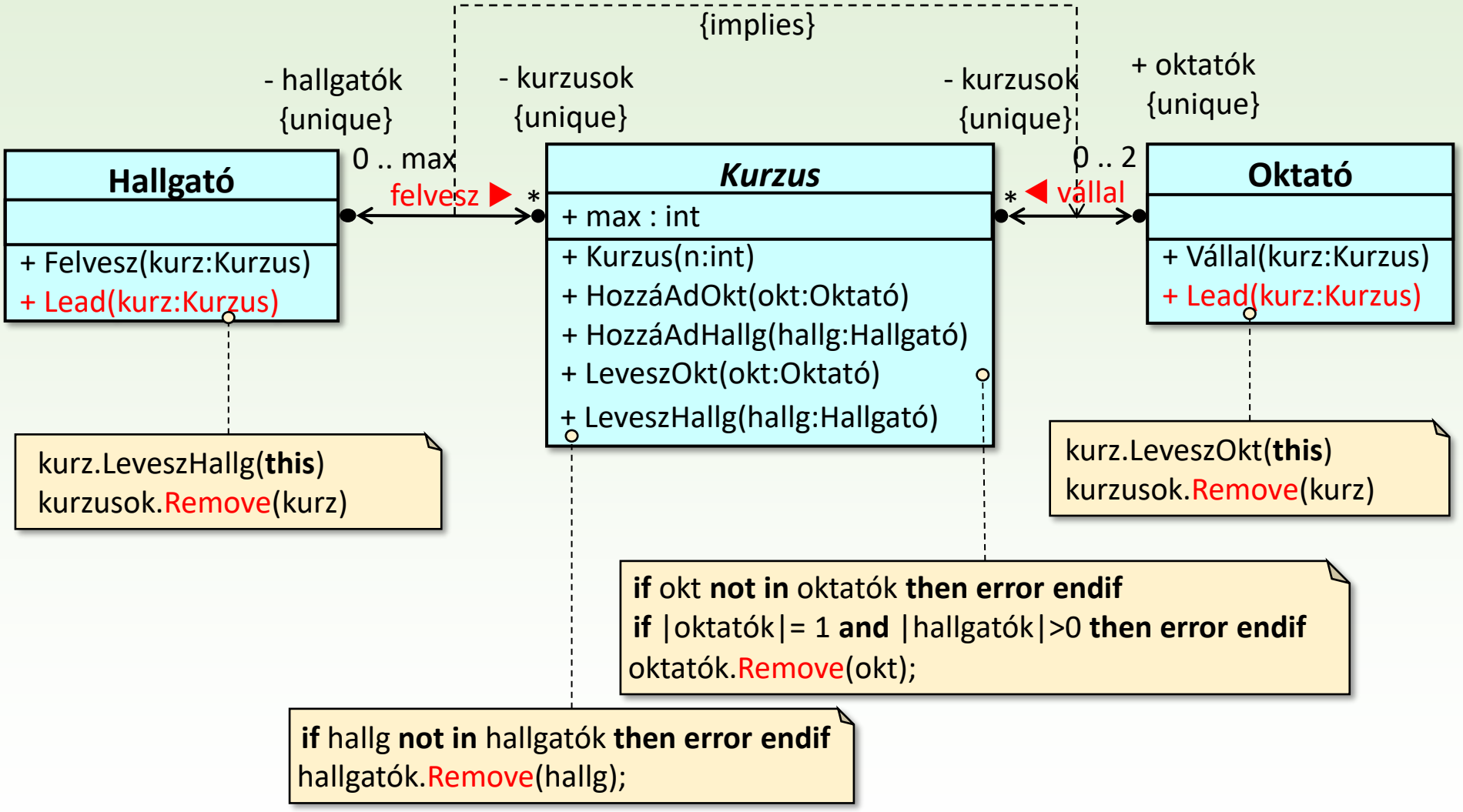

```
public int Max { get; private set; }
private readonly List<Student> students = new (); // 0 .. max
private readonly List<Teacher> teachers = new (); // 0 .. 2

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

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

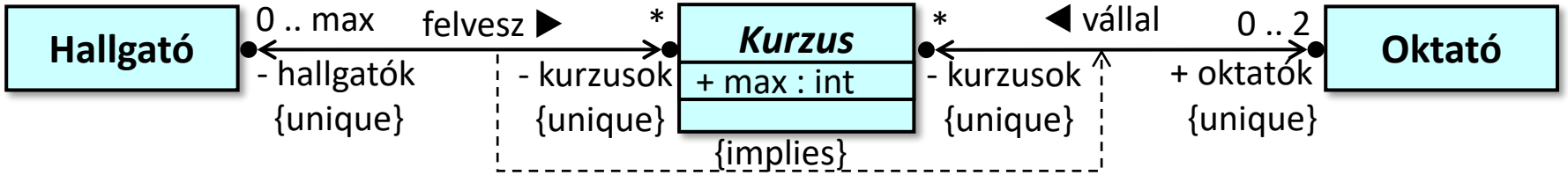
class Course

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

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

class Student

Kapcsolatok bontása






```
private readonly List<Course> courses
                  = new ();
public void SignsUp(Course c) { ... }
public void SignsDown(Course c)
{
   c.RemoveStudent(this);
   courses.Remove(c);
}
```

class Student

```
private readonly List<Course> courses
                  = new ();
public void Undertakes(Course c) { ... }
public void Drops(Course c)
{
   c.RemoveTeacher(this);
   courses.Remove(c);
}
```

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.Contains(t)) throw new TeacherNotFoundInCourse();
   if (teachers.Count == 1 && students.Count > 0) throw new TeacherCannotRemoved();
   teachers.Remove(t);
}

public void RemoveStudent(Student s)
{
   if (!students.Contains(s)) throw new StudentNotFoundInCourse();
   students.Remove(s);
}
```

class Course

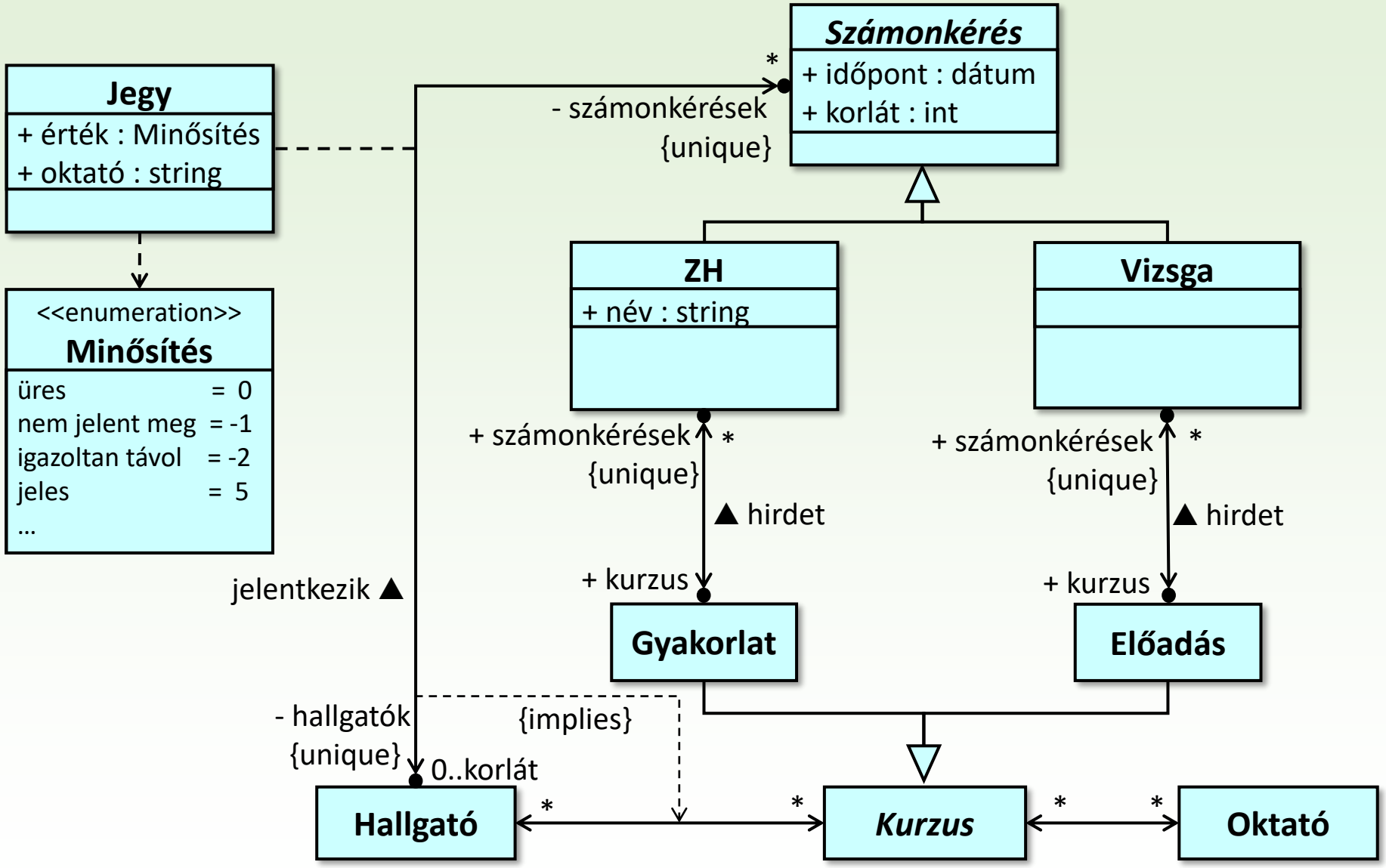
3.rész
Kurzus-számonkérés-hallgató kapcsolatok

Gregorics Tibor

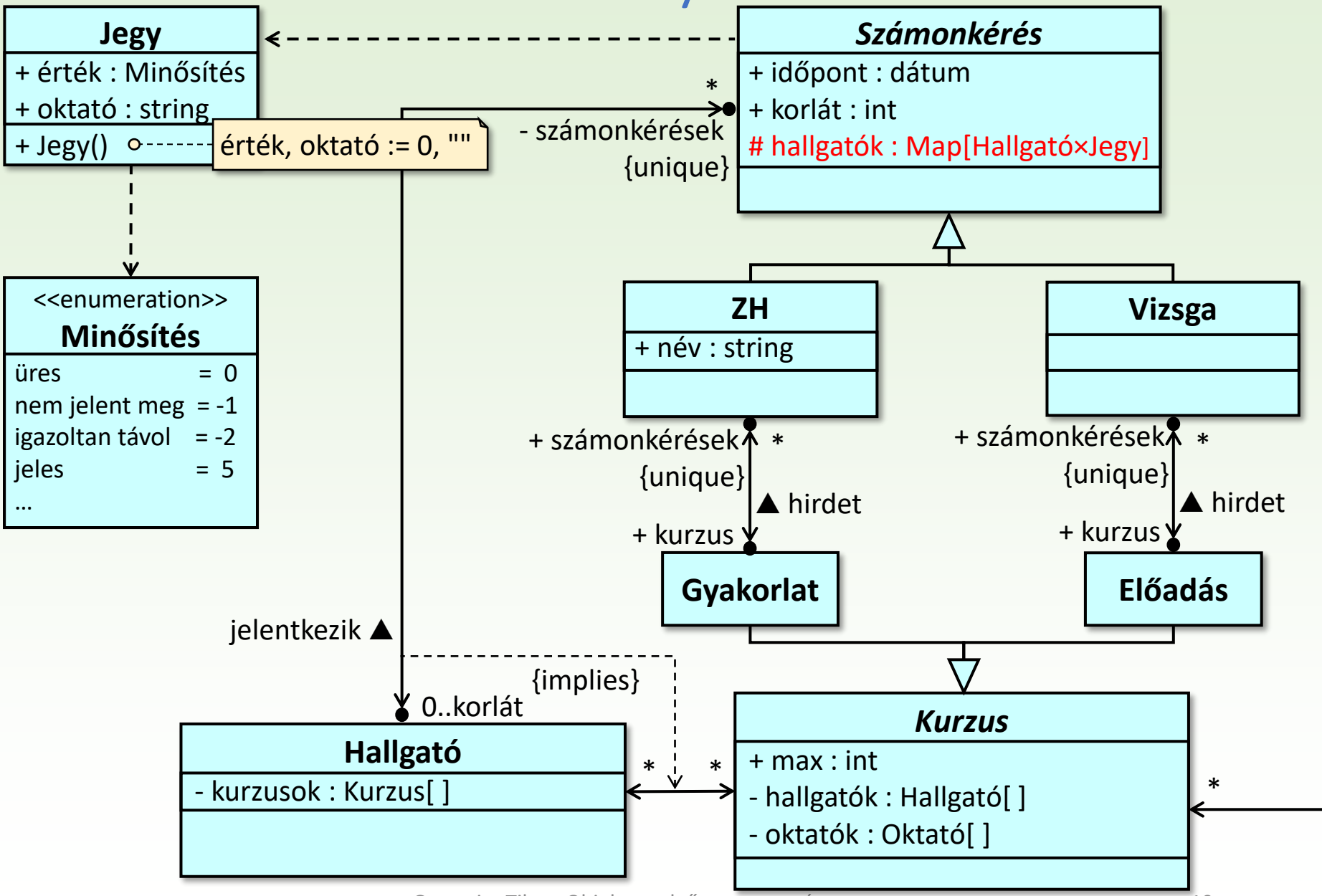
gt@inf.elte.hu

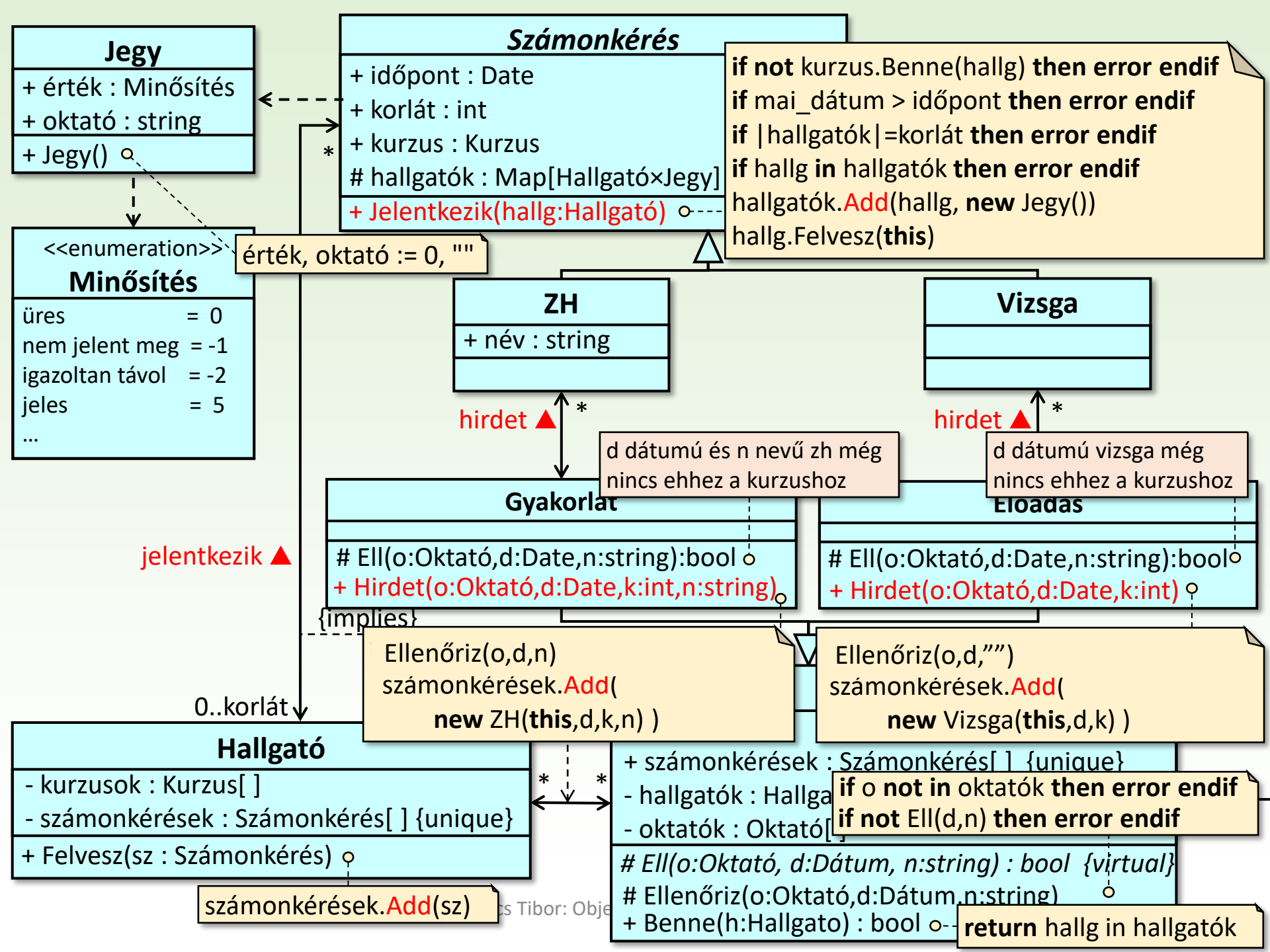
http://people.inf.elte.hu/gt/oep

Számonkérések osztályai



Asszociációs osztály eliminálása




Jegy
+ érték : Minősítés
+ oktató : string
+ Jegy()
Számonkérés
+ időpont : Date
+ korlát : int
+ kurzus : Kurzus
# hallgatók : Map[Hallgató×Jegy]
+ Jelentkezik(hallg:Hallgató)
if not kurzus.Benne(hallg) then error endif
if mai_dátum > időpont then error endif
if |hallgatók|=korlát then error endif
if hallg in hallgatók then error endif
hallgatók.Add(hallg, new Jegy())
hallg.Felvesz(this)
*
<<enumeration>>
Minősítés
üres = 0
nem jelent meg = -1
igazoltan távol = -2
jeles = 5
...
érték, oktató := 0, ""
ZH
+ név : string
Vizsga
hirdet
*
hirdet
*
d dátumú és n nevű zh még nincs ehhez a kurzushoz
d dátumú vizsga még nincs ehhez a kurzushoz
Gyakorlat
Ell(o:Oktató,d:Date,n:string):bool
+ Hirdet(o:Oktató,d:Date,k:int,n:string)
Előadás
Ell(o:Oktató,d:Date,n:string):bool
+ Hirdet(o:Oktató,d:Date,k:int)
jelentkezik
{implies}
Ellenőriz(o,d,n)
számonkérések.Add(
new ZH(this,d,k,n))
Ellenőriz(o,d,"")
számonkérések.Add(
new Vizsga(this,d,k))
0..korlát
Hallgató
- kurzusok : Kurzus[ ]
- számonkérések : Számonkérés[ ] {unique}
+ Felvesz(sz : Számonkérés)
számonkérések.Add(sz)
* *
+ számonkérések : Számonkérés[ ] {unique}
- hallgatók : Hallga
- oktatók : Oktató[
if o not in oktatók then error endif
if not Ell(d,n) then error endif
Ell(o:Oktató, d:Dátum, n:string) : bool {virtual}
Ellenőriz(o:Oktató,d:Dátum,n:string)
+ Benne(h:Hallgato) : bool
return hallg in hallgatók

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;
}

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

erre a tesztkörnyezetnek (főprogram) lesz szüksége

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

C# kód

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

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

class Control

Dictionary metódusa

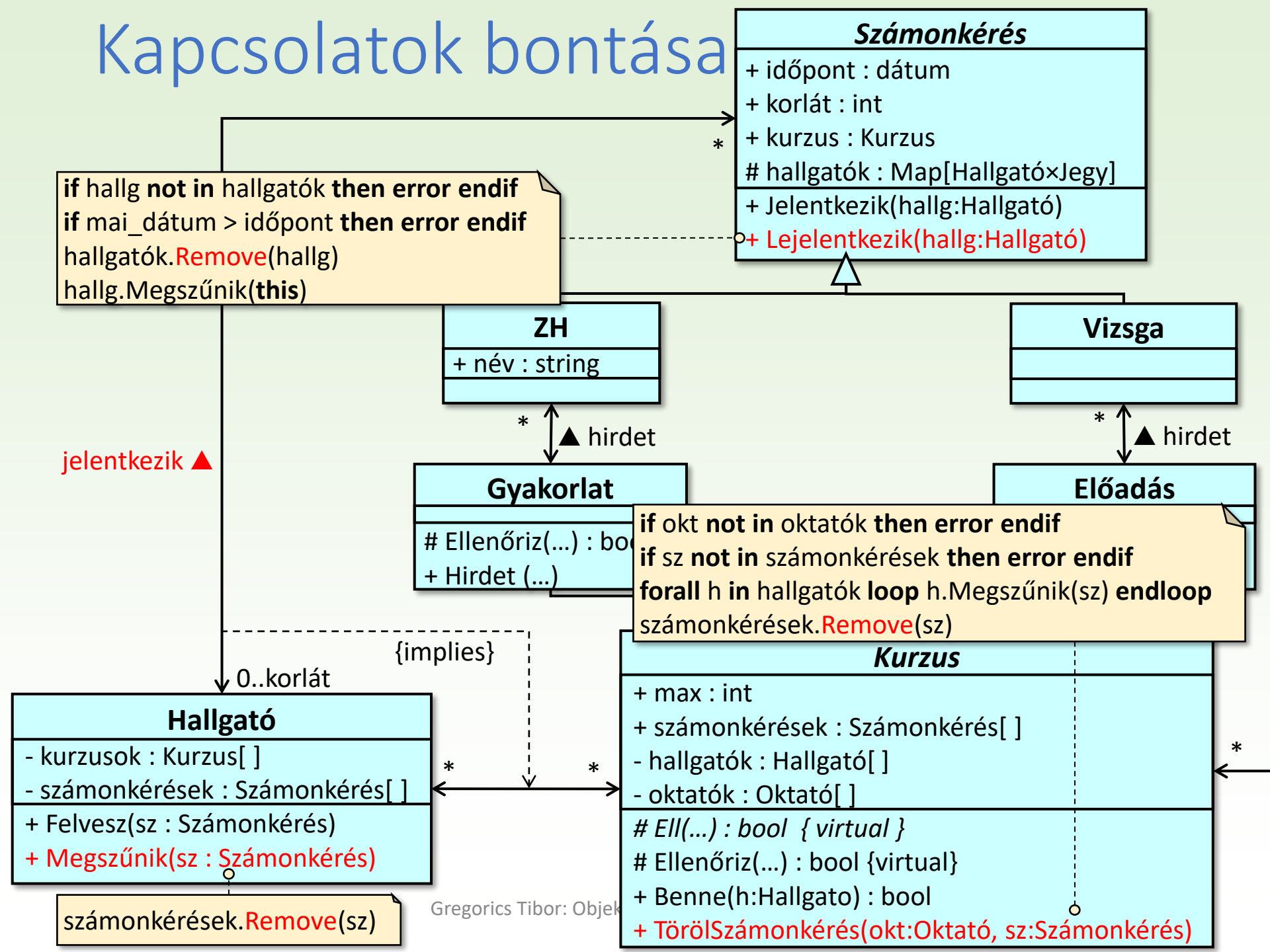
```
public void AddControl(Control ex)
{
   controls.Add(ex);
}
```

class Student : Person

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

class Course

Kapcsolatok bontása



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.RemoveControl(control);
   controls.Remove(control);
}
```

class Course

```
protected readonly Dictionary<Student, Grade> students = new ();
public void Unregistrate(Student student)
{
   if (!students.ContainsKey(student)) throw new StudentNotRegistrated();
   if (DateTime.Now > date ) throw new TimeLimit();
   students.Remove(student);
   student.RemoveControl(this);
}
```

class Control

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

class Student : Person

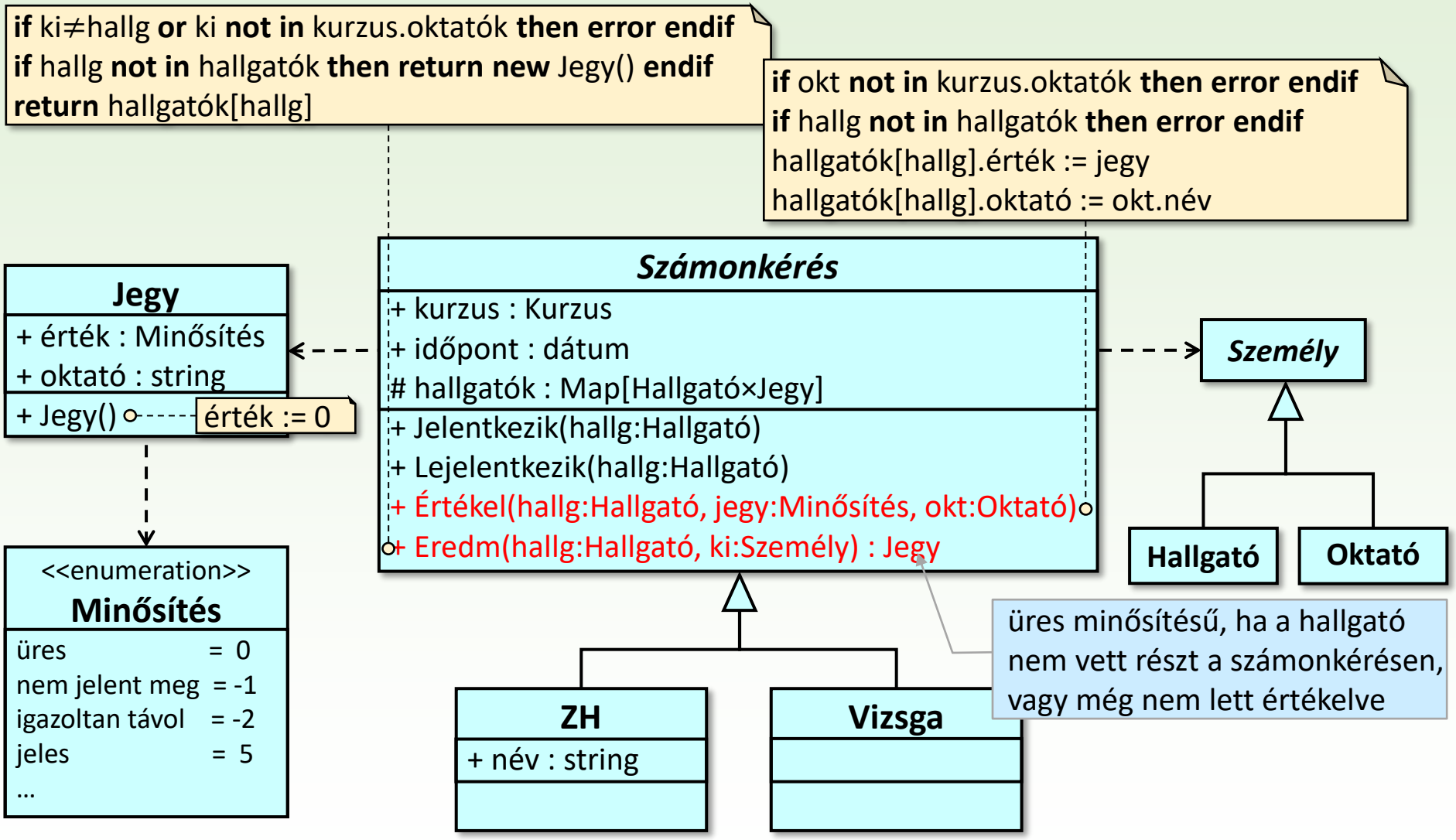
4.rész
Számonkérések értékelése, megtekintése

Gregorics Tibor

gt@inf.elte.hu

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

Értékelés és megtekintése



C# kód

kurzus oktatói listájának getter-e

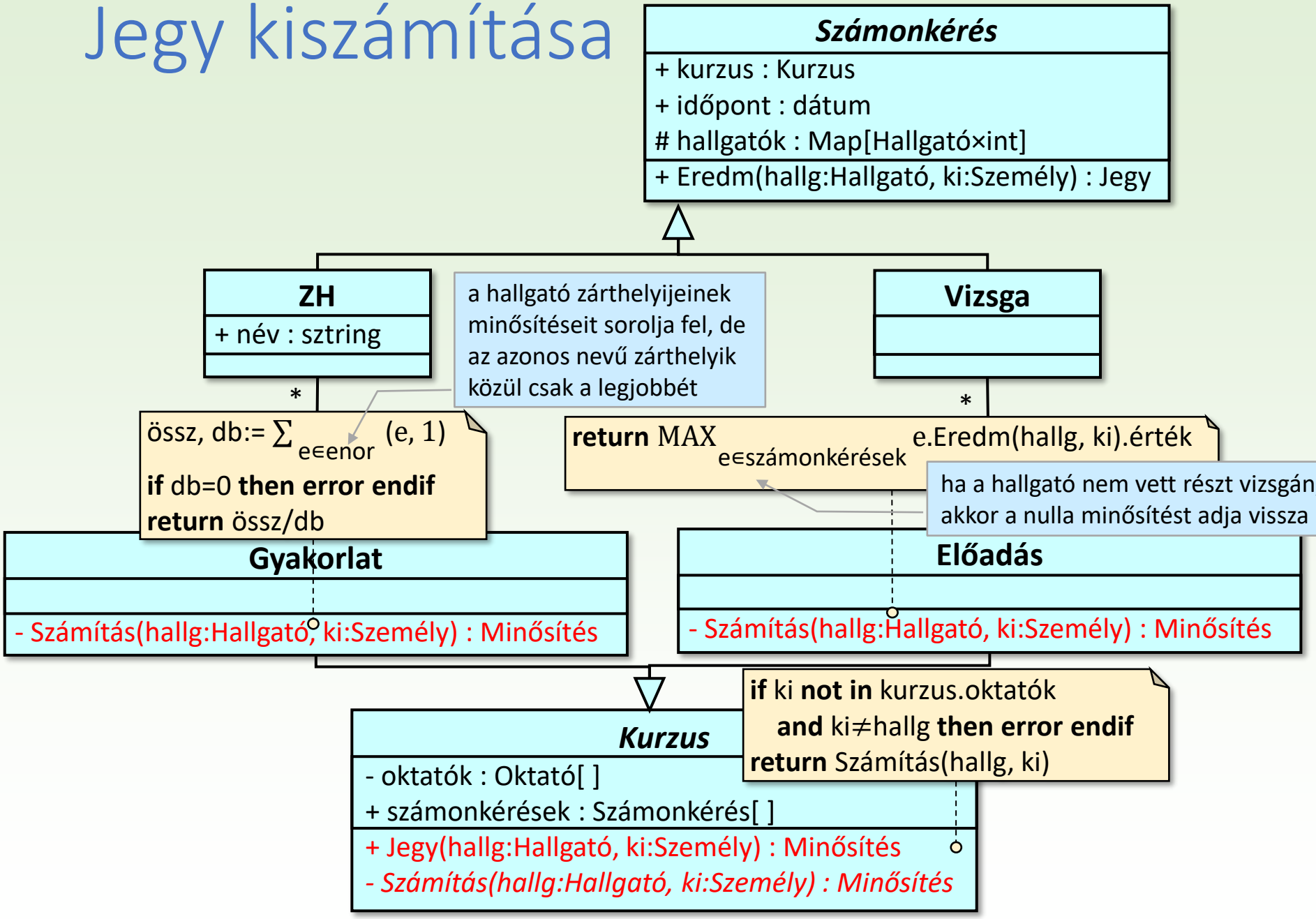
```
public void Evaluate(Student student, Category category, Teacher teacher)
{
   if (!course.Teachers.Contains(teacher)) throw new Course.Teacher FoundInCourse();
   if (!students.ContainsKey(student)) throw new StudentNotRegistrated();
   students[student].value = category;
   students[student].teacher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



Lekérdezések

```
enum Category { excellent = 5, good = 4, satisfactory = 3, poor = 2,
                insufficient = 1, no_grade = 0, not_appear = -1 }
class Grade
{
   public Category value;
   public string teacher;
   public Grade() { value = Category.no_grade; teacher = "" ; }
   public override string ToString()
   {
      return value.ToString() + " by " + teacher;
   }
}
```

ToString() felülírása

class Course

```
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);
```

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)));
}
```

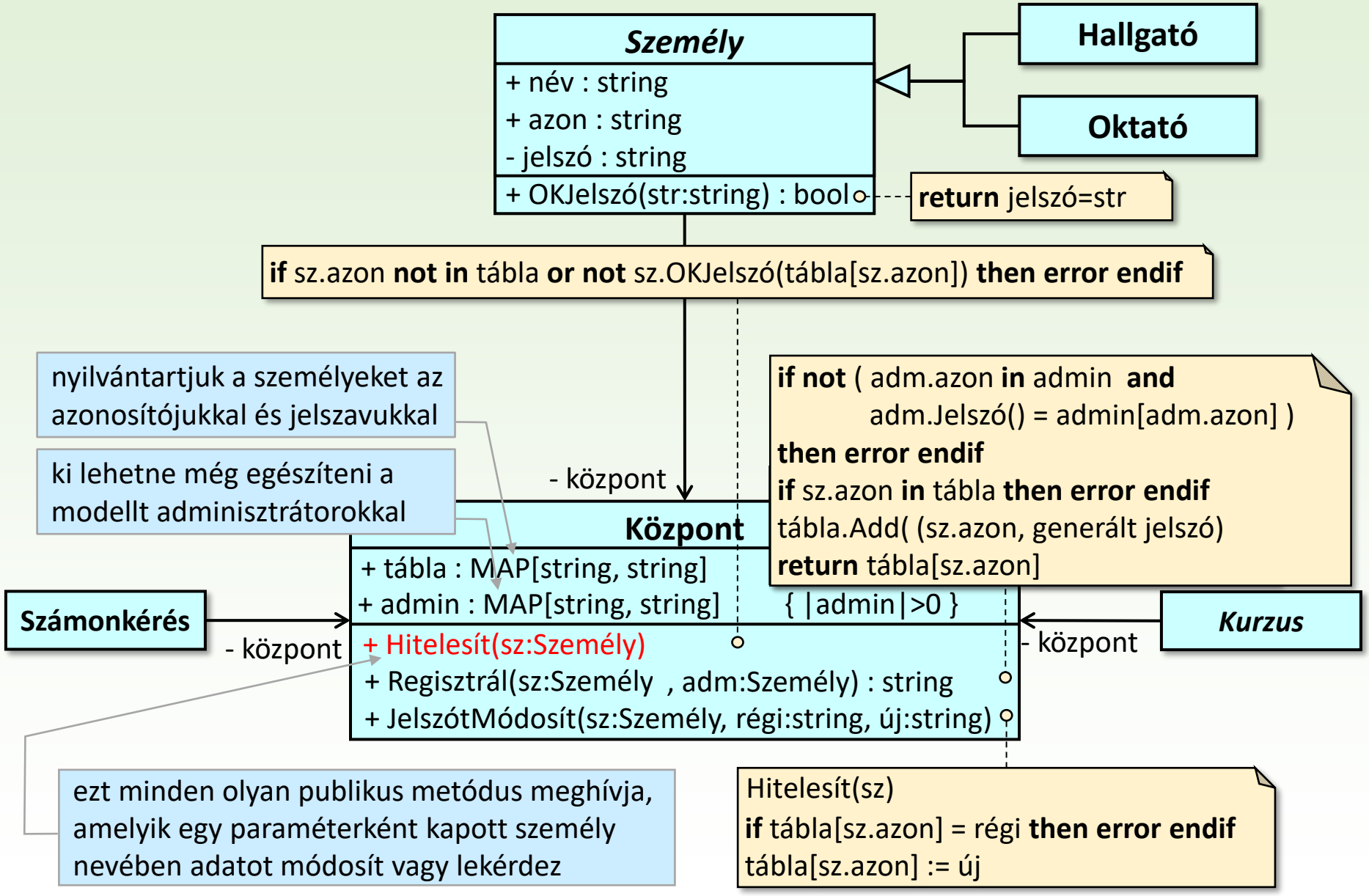
a minősítések számértékével kell dolgozni

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
```

Hitelesítés



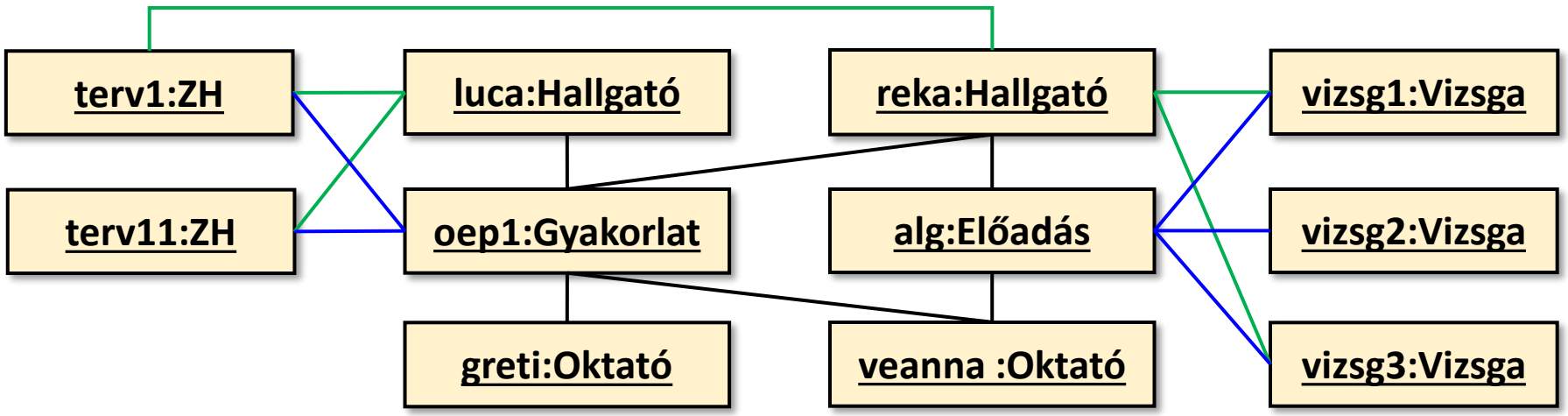
5.rész
Modell tesztelése

Gregorics Tibor

gt@inf.elte.hu

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

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", "CCCCCC");
```

```
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 terv11 = 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.
Exam exam3  = alg.Announce(veanna, DateTime.Parse("2023.
```

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

Egység tesztek tervezése

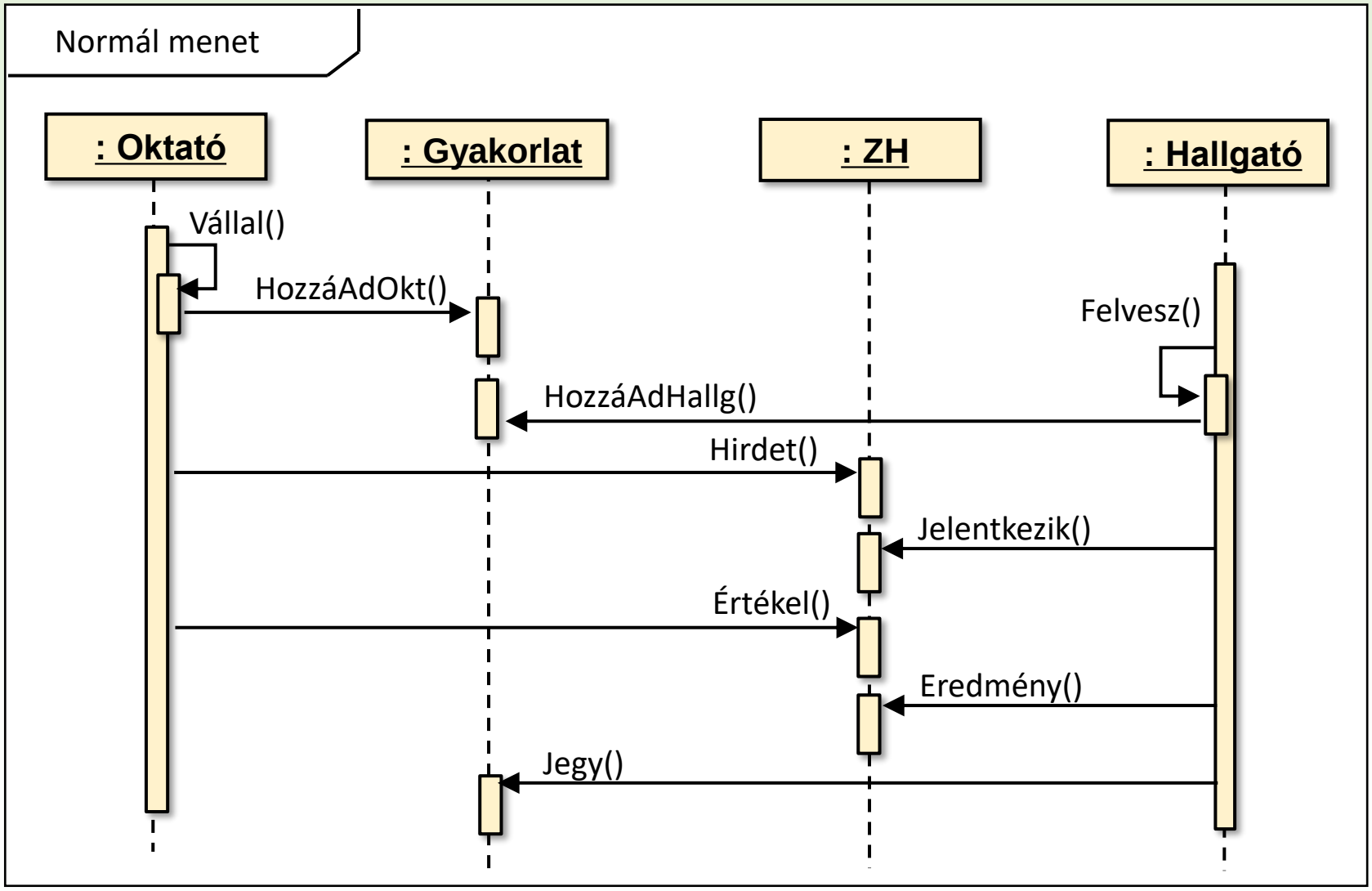
Például az Eredm() metódus egy számonkérésnek korábban rögzített értékelését adja vissza a javító oktató nevével, de (0,””), ha a hallgató nem vett részt a számonkérésen (0,””), ha még nem lett értékelve a hallgató munkája

- 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.

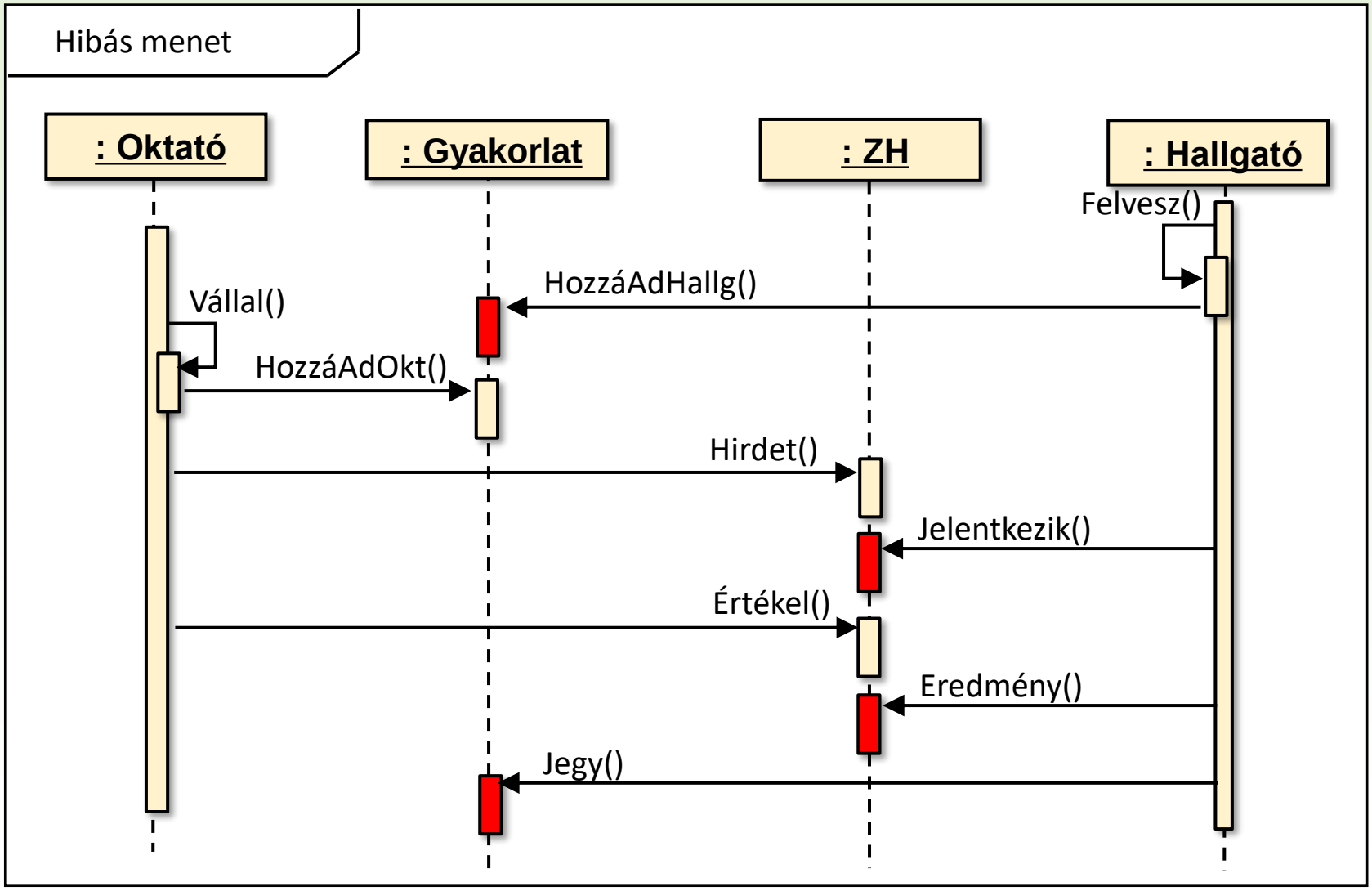
Például Jegy() metódusa

- Ha azonban egy metódus tevékenysége nevezetes algoritmus mintára épül (ilyen az érdemjegy kiszámítása), akkor az algoritmus mintára jellemző szürke doboz teszteseteket is próbáljuk ki.
- A metódusokban végzett hibaellenőrzés kivétel-dobásokat generál, amelyek mindegyikét ki kell váltanunk egy megfelelő adattal.
- 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.

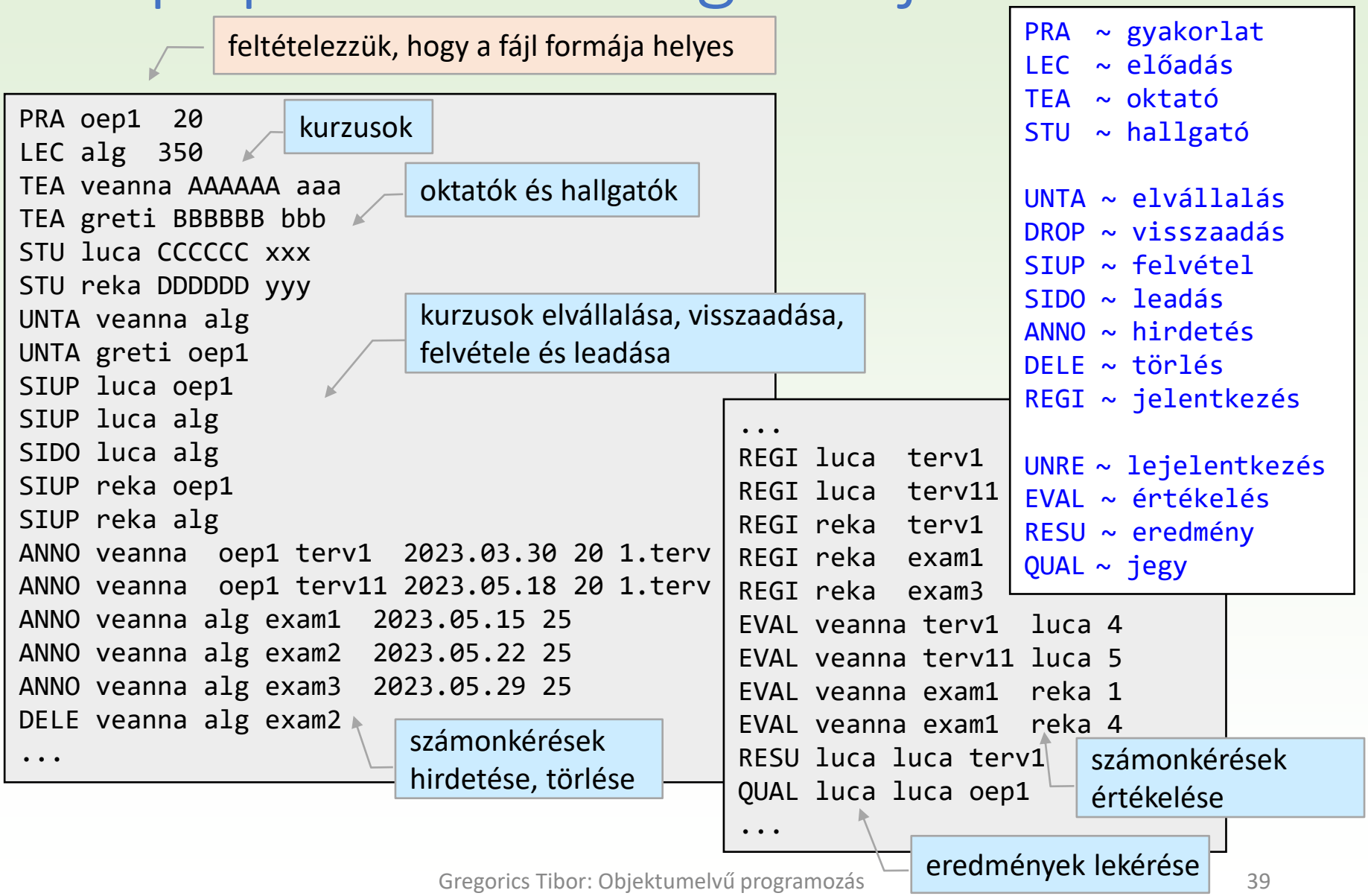
Integrációs tesztek tervezése



Integrációs tesztek tervezése



Felpopulálás szöveges fájlból



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.

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

```
...
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]]).SignsUp(Course(tokens[2])); break;
case "SIDO": ((Student)persons[tokens[1]]
case "ANNO": Course course = Course(toker
             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;
```