Objektumorientált programozás

Öröklés

Goldschmidt Balázs

balage@iit.bme.hu

Ismétlés

Egyetemi nyilvántartás: ismétlés

- A feladat: *egyetemi nyilvántartás*
  - hallgatók
    - név, neptunkód, szül. év, átlag, megsz. kreditek
  - oktatók
    - név, neptunkód, szül. év, beosztás
  - kurzusok
    - név, neptun, ki oktatja, ki hallgatja

Hallgató

```
public class Student {
  private String name; private String neptun;
  private int yob; // Year Of Birth
  private double average;
  private int credits;

  public Student(String na, String ne, int y) {
    name = na; neptun = ne; yob = y;
    average = 0.0; credits = 0;
  }

  public String getName() { return name; }
  public String getNeptun() { return neptun; }
  ...
```

Hallgató

```
  ...

  public void addMark(int mark, int credit) {
    average = (average*credits + mark*credit) /
              (credits+credit);
    credits += credit;
  }
  public String toString() {
    return name+" ("+neptun+") "
           +yob+", "+average+", "+mark;
  }
}
```

Oktató

```
public class Teacher {
  private String name; private String neptun;
  private int yob; // Year Of Birth
  private String title;

  public Teacher(String na, String ne, int y) {
    name = na; neptun = ne; yob = y;
    title = "assistant teacher";
  }

  public String getName() { return name; }
  public String getNeptun() { return neptun; }
  ...
```

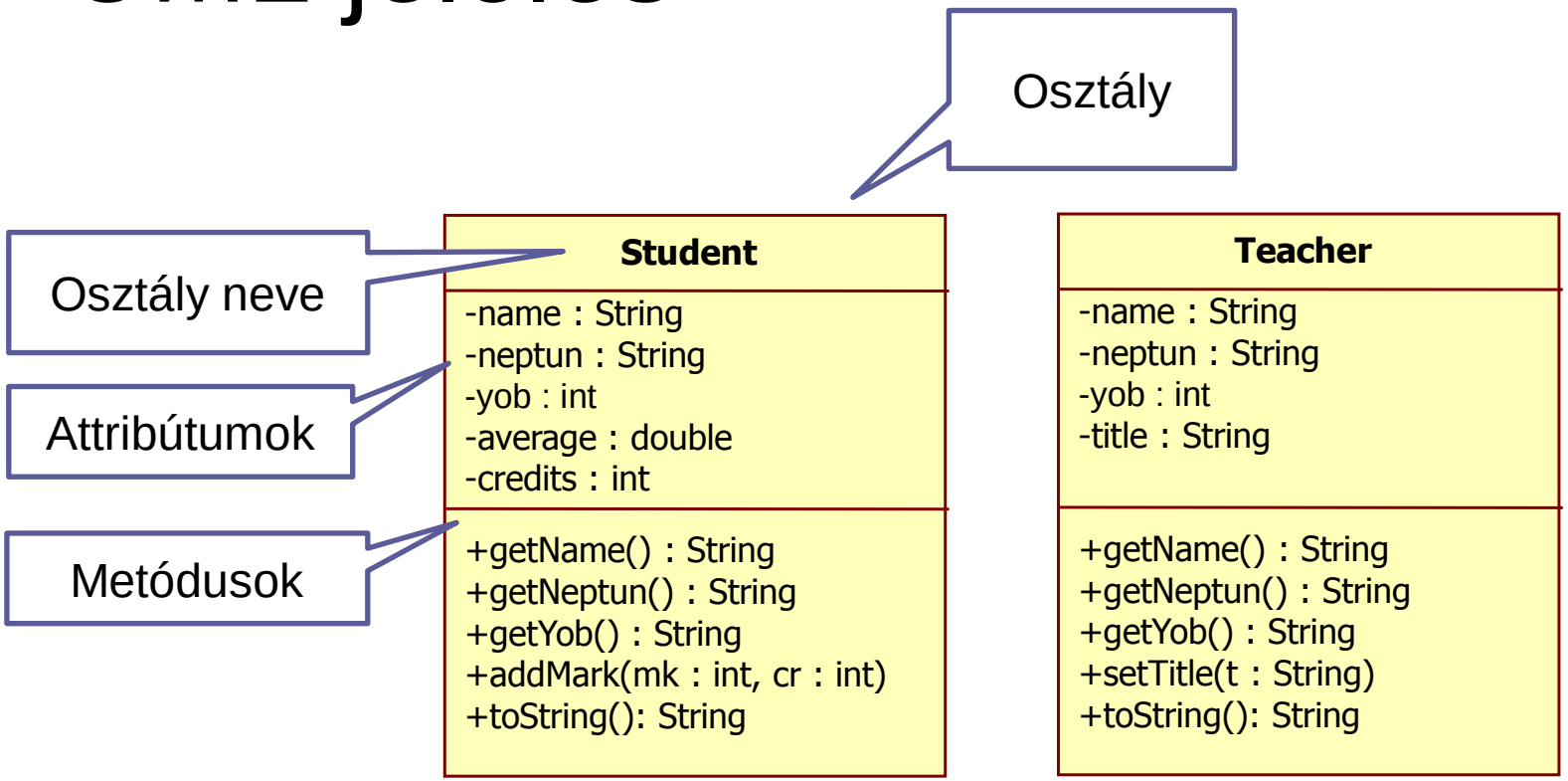
Oktató

```
  ...

  public void setTitle(String s) {
       title = s;
  }
  public String toString() {
    return name+" ("+neptun+") "
           +yob+", "+title;
  }
}
```

Példa

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Student s2 = new Student("Nagy Károly", "XXX111", 1998);
Student s3 = new Student("Kis Pippin", "111XXX", 1999);

s1.addMark(4, 2);

Teacher t1 = new Teacher("Rend Elek", "Q1W2E3", 1973);
Teacher t2 = new Teacher("Csirke Béla", "OKTAT6", 1980);

System.out.println(s1);
System.out.println(s2);
System.out.println(s3);

System.out.println(t1);
System.out.println(t2);
```

UML jelölés


Osztály
Osztály neve
Attribútumok
Metódusok
Student
-name : String
-neptun : String
-yob : int
-average : double
-credits : int
+getName() : String
+getNeptun() : String
+getYob() : String
+addMark(mk : int, cr : int)
+toString(): String
Teacher
-name : String
-neptun : String
-yob : int
-title : String
+getName() : String
+getNeptun() : String
+getYob() : String
+setTitle(t : String)
+toString(): String

Öröklés

Lehet-e egyszerűbben?

- Hallgató és oktató hasonló tartalmú
  - *name*, *neptun*, *yob* ugyanaz
  - ezekhez hasonló getterek tartoznak
- Vonjuk össze:
  - Legyen egy *Person*, amiben van
    - *name*, *neptun*, *yob*
    - *getter* és *toString* a fentiekhez
    - konstruktor a létrehozáshoz

DRY: don't repeat yourself

Person

```
public class Person {
  private String name; private String neptun;
  private int yob; // Year Of Birth

  public Person(String na, String ne, int y) {
    name = na; neptun = ne; yob = y;
  }

  public String getName() { return name; }
  public String getNeptun() { return neptun; }
  public int getYob() { return yob; }
  public String toString() {
    return name+" ("+neptun+") "+yob;
  }
}
```

Person a Hallgató őse (rossz)

```
public class Student extends Person {
  private double average;
  private int credits;
  public Student(String na, String ne, int y) {
    name = na; neptun = ne; yob = y;
    average = 0.0; credits = 0;
  }
  public void addMark(int mark, int credit) {
    average = (average*credits + mark*credit) /
              (credits+credit);
    credits += credit;
  }
  public String toString() {
    return super.toString() +", "+average+", "+credits;
  }
}
```

Ős megadása

Ős attribútumai privátok!!!

Ős metódusa

Person a Hallgató őse (jó)

```
public class Student extends Person {
  private double average;
  private int credits;
  public Student(String na, String ne, int y) {
    super(na, ne, y);
    average = 0.0; credits = 0;
  }
  public void addMark(int mark, int credit) {
    average = (average*credits + mark*credit) /
              (credits+credit);
    credits += credit;
  }
  public String toString() {
    return super.toString() +", "+average+", "+credits;
  }
}
```

Ős megadása

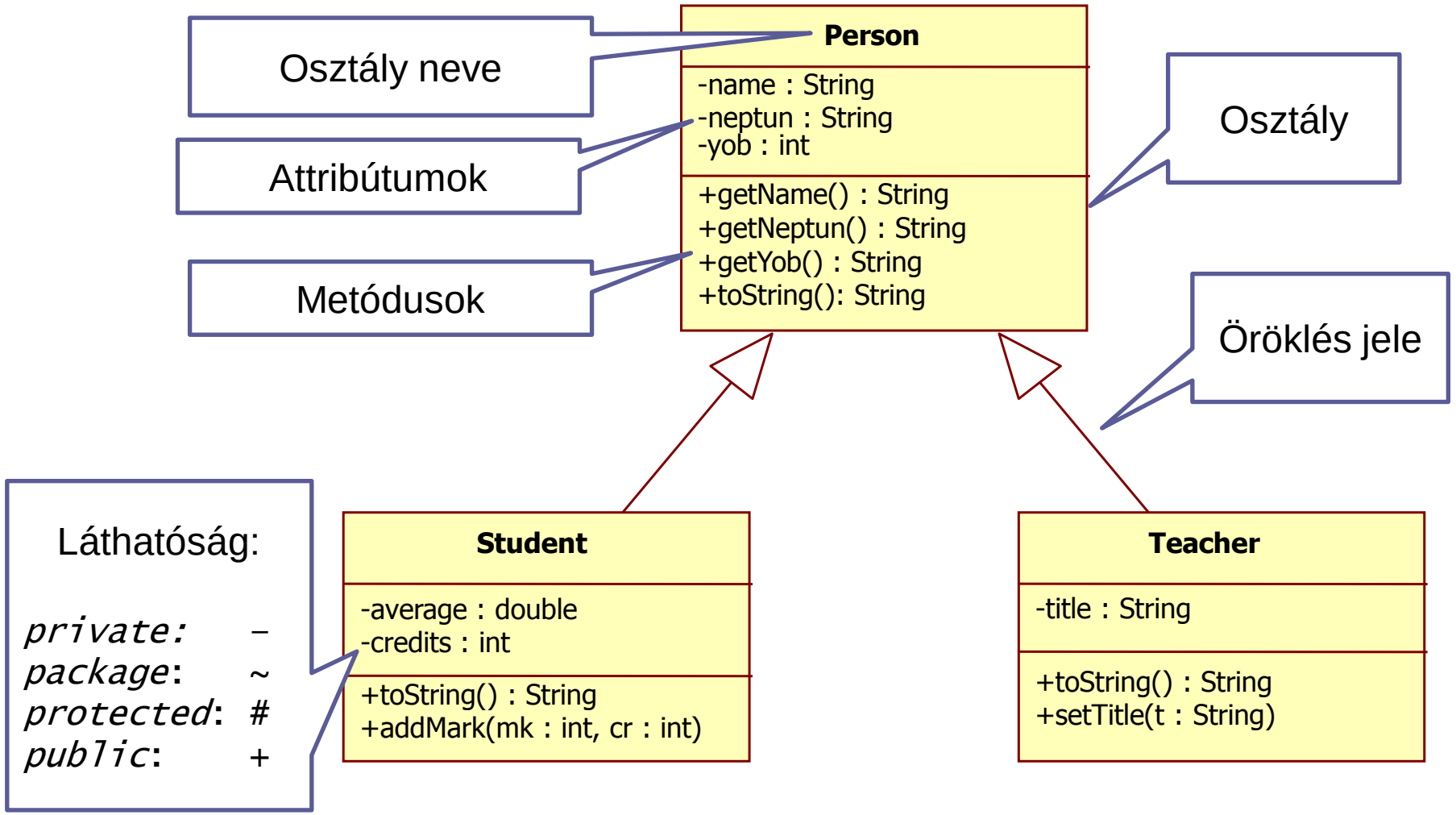
Ős konstruktora

Ős metódusa

Person az Oktató őse

```
public class Teacher extends Person {
  private String title;

  public Teacher(String na, String ne, int y) {
    super(na,ne,y);
    title = "assistant teacher";
  }
  public void setTitle(String s) {
       title = s;
  }
  public String toString() {
    return super.toString() +", "+title;
  }
}
```

UML: Person és gyerekei



Példa újra

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Student s2 = new Student("Nagy Károly", "XXX111", 1998);
Student s3 = new Student("Kis Pippin", "111XXX", 1999);

s1.addMark(4, 2);

Teacher t1 = new Teacher("Rend Elek", "Q1W2E3", 1973);
Teacher t2 = new Teacher("Csirke Béla", "OKTAT6", 1980);

System.out.println(s1);
System.out.println(s2);
System.out.println(s3);

System.out.println(t1);
System.out.println(t2);
```

Öröklés szabályai

- Őstől minden öröklődik
- Egyetlen közvetlen ős
  - közvetve lehet több...
- Ős elemeit közvetlenül *super* kulcsszóval
  - konstruktor mint metódushívás
    - ha nem hívjuk, a default hívódik (ilyenkor hiba, ha nincs def.)
  - mezők mint mezőelérés
- Ahol őst várnak, leszármazott is jöhet
  - ún. **Liskov helyettesítési elv** *(Liskov Substitution Principle)*, konform öröklés, kompatibilitás

Öröklődés és a láthatóság

- Private
  - csak az ősből elérhető
  - pl. ősből örökölt metódus eléri
- Package (nincs jelölése)
  - azonos csomagbeli eléri
- Protected
  - leszármazott és azonos csomagbeli eléri
- Public
  - mindenki eléri

Öröklés és láthatóság, példa 1

- Legyen testsúly!

```
public class Person {
  // ...
  private double weight; // testsúly, privát
  public double getWeight() { return weight; }
  public void setWeight(double w) { // setter, publikus
    weight = w;
  }
  ...
```

- Következmények
  - Student tudja módosítani
  - Bárki tudja módosítani

Öröklés és láthatóság, példa 2

- Legyen testsúly!

```
public class Person {
  // ...
  private double weight; // testsúly, privát
  public double getWeight() { return weight; }
  protected void setWeight(double w) { //setter, prot.
    weight = w;
  }
  ...
```

- Következmények
  - *Student*, *Teacher* tudja módosítani
  - más nem fér hozzá

Statikus és dinamikus típus

- Mi történik itt?

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Teacher t1 = new Teacher("Rend Elek", "Q1W2E3", 1973);
Person  p1 = new Person("Nagy Károly", "XXX111", 1998);

Person p2 = s1;
Person p3 = t1;

System.out.println(s1); //Gipsz Jakab (1A2B3C), 1996, 0.0, 0
System.out.println(t1); //Rend Elek (Q1W2E3), 1973, assis...
System.out.println(p1); //Nagy Károly (XXX111), 1998
System.out.println(p2); //Gipsz Jakab (1A2B3C), 1996, 0.0, 0
System.out.println(p3); //Rend Elek (Q1W2E3), 1973, assis...
```

dinamikus típus

statikus típus

Teacher *toString()*-je!!!

Statikus vs dinamikus típus

- Minden objektum-változónak van
  - statikus típusa:
    - a deklarálásakor adjuk meg
    - soha nem változik (statikus)
    - fordításkor eldől
  - dinamikus típusa
    - értékadáskor adjuk meg
    - a referált objektum valós típusával egyezik meg
    - értékadáskor megváltozhat (dinamikus)
    - futáskor dől el

Statikus és dinamikus típus

- Mi történik itt?

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Teacher t1 = new Teacher("Rend Elek", "Q1W2E3", 1973);
Person  p1 = new Person("Nagy Károly", "XXX111", 1998);

Person p2 = s1;
Person p3 = t1;

System.out.println(s1); //Gipsz Jakab (1A2B3C), 1996, 0.0, 0
System.out.println(t1); //Rend Elek (Q1W2E3), 1973, assis...
System.out.println(p1); //Nagy Károly (XXX111), 1998
System.out.println(p2); //Gipsz Jakab (1A2B3C), 1996, 0.0, 0
System.out.println(p3); //Rend Elek (Q1W2E3), 1973, assis...
```

dinamikus típus

statikus típus

dinamikus típus *toString()*-je!!!

Virtuális metódusok

- Minden, nem privát metódus virtuális
  - ha meghívjuk, a dinamikus típusban definiált változata fut le
  - ha ilyen nincs, akkor az öröklési hierarchiában egyre feljebb keresünk
- Ha az ős metódusát kell hívni, akkor használjunk *super*-t

```
public String toString() {
    return super.toString() +", "+title;
}
```

Metódushívás-szabály

- A statikus típus metódusai hívhatók
  - a fordító ellenőrzi, van-e neki
- A dinamikus típus metódusa fut (ha van)
  - futás közben a valódi típust használja
- *this* típusa
  - *statikus*: amelyik osztályban deklaráltuk a metódust
  - *dinamikus*: amelyik osztály példányán meghívtuk a metódust

Metódushívás-szabály példa

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Person  p1 = new Person("Nagy Károly", "XXX111", 1998);
Person  p2 = s1;

System.out.println(s1); //Gipsz Jakab (1A2B3C), 1996, 0.0, 0
System.out.println(p1); //Nagy Károly (XXX111), 1998
System.out.println(p2); //Gipsz Jakab (1A2B3C), 1996, 0.0, 0
s1.addMark(4,2); // OK: s1 statikus típusában van addMark
p1.addMark(4,2); // HIBA: p1 statikus típusában nincs ilyen
p2.addMark(4,2); // HIBA: p2 statikus típusában nincs ilyen
                 // pedig p2 dinamikus típusa tudná
```

Paraméterátadás és öröklés

```
public void foo(Person p) {
  p.addMark(4,2); // HIBA: p statikus típusában nincs
  System.out.println(p); // OK: kiírja p.toString()-et
}
public void bar(Student p) {
  p.addMark(4,2); // OK: p statikus típusában van
  System.out.println(p); // OK: kiírja p.toString()-et
}
```

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Person  p1 = new Person("Nagy Károly", "XXX111", 1998);

foo(p1); // OK: p1 Person
foo(s1); // OK: s1 lehet Person, mert az az őse
bar(p1); // HIBA: p1 nem Student
bar(s1); // OK: s1 Student
```

Paraméterátadás és öröklés

```
public void foo(Person p) {
  p.addMark(4,2); // HIBA: p statikus típusában nincs
  System.out.println(p); // OK: kiírja p.toString()-et
}
public void bar(Student p) {
  p.addMark(4,2); // OK: p statikus típusában van
  System.out.println(p); // OK: kiírja p.toString()-et
}
```

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Person  p1 = s1;

foo(s1); // OK: s1 lehet Person, mert az az őse
foo(p1); // OK: p1 statikus típusa Person
bar(s1); // OK: s1 Student
bar(p1); // HIBA: p1 statikus típusa nem Student!!!
bar((Student)p1)); // OK: cast-oltuk Studentre
```

ClassCastException, ha nem lehet

Absztrakt osztályok

Absztrakt osztályok

- Person létezhet?
  - mit tehetünk, ha nem akarjuk engedni, hogy *Person* önmagában létezzen
    - csak *Student* és *Teacher* lehet
    - maradjon meg az örökléssel járó előny
- Ha nem: legyen *abstract*
  - nem példányosítható
  - statikus típus lehet
  - formális paraméter típusa lehet

Abstract Person

```
abstract public class Person {
  private String name; private String neptun;
  private int yob; // Year Of Birth

  public Person(String na, String ne, int y) {
    name = na; neptun = ne; yob = y;
  }

  public String getName() { return name; }
  public String getNeptun() { return neptun; }
  public int getYob() { return yob; }
  public String toString() {
    return name+" ("+neptun+") "+yob;
  }
}
```

Abstract metódus

- Legyen egy köszönés metódus!
  - `public String greetings()`
  - Hallgatónál: "Helló!"
  - Oktatónál: "Üdvözletem!"
  - Personnál: ???
- Elvárás
  - legyen Person típuson hívható
  - ne legyen Person-ban megírva
    →*abstract* metódus

TDA: tell, don't ask

Abstract metódus

```
abstract public class Person {
  private String name; private String neptun;
  private int yob; // Year Of Birth

  public Person(String na, String ne, int y) {
    name = na; neptun = ne; yob = y;
  }
  ...
  abstract public String greetings();

}
```

Person greetings-szel

```
public class Student extends Person {
   ...
   public String greetings() {
      return "Helló!";
   }

}
```

Felülírja az ős nem definiált metódusát

```
public class Teacher extends Person {
   ...
   public String greetings() {
      return "Üdvözletem!";
   }

}
```

Felülírja az ős nem definiált metódusát

Statikus és dinamikus típus

- Abstract metódussal

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Teacher t1 = new Teacher("Rend Elek", "Q1W2E3", 1973);
Person  p1 = new Person("Nagy Károly", "XXX111", 1998);

Person p2 = s1;
Person p3 = t1;

System.out.println(s1.greetings()); // Helló
System.out.println(t1.greetings()); // Üdvözletem
System.out.println(p2.greetings()); // Helló
System.out.println(p3.greetings()); // Üdvözletem
```

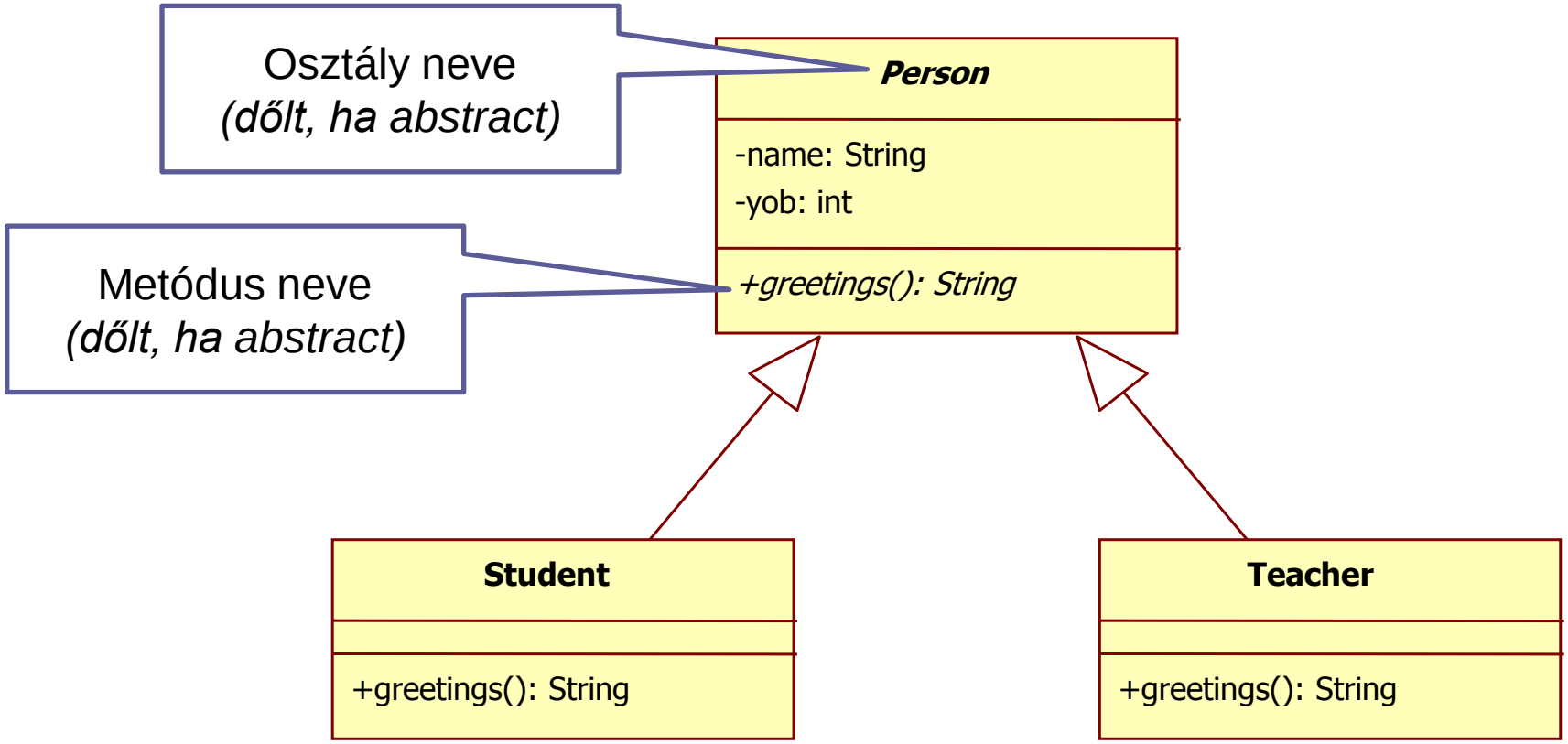
Nem példányosítható

Paraméterátadás és öröklés

```
public void foo(Person p) {
  System.out.println(p); // OK: kiírja p.toString()-et
}
public void bar(Student p) {
  p.addMark(4,2); // OK: p statikus típusában van
  System.out.println(p); // OK: kiírja p.toString()-et
}
```

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Person  p1 = new Person("Nagy Károly", "XXX111", 1998);
Person  p1 = s1; // statikus típus lehet

foo(p1); // OK: p1 Person
foo(s1); // OK: s1 lehet Person, mert az az őse
bar(s1); // OK: s1 Student
```

UML jelölés


Osztály neve
(dőlt, ha abstract)
Metódus neve
(dőlt, ha abstract)
Person
-name: String
-yob: int
+greetings(): String
Student
+greetings(): String
Teacher
+greetings(): String

Kivételkezelés és öröklés

Öröklés: kivételek

```
...
Course c = new Course("Zabhegyezés", "BMEVIIIZZ00", 24);
Teacher t = new Teacher("Vastagh Béla", "VSTGBL",1975);
c.setTeacher(t);
try {
  c.addStudent(new Student("Lutz Ernő", "LTZRN0",1996));
  c.addStudent(new Student("Szőke Barna", "BRN123",1996));
  c.addStudent(new Student("Hervadt Virág", "HRVDTV",1995));
} catch (Exception e) {
  System.err.println(e.getMessage());
  // e.printStackTrace();
}
...
```

Kivételkezelés szabályai

- El nem kapott kivétel típusát jelezni kell a metódus fejlécében (*throws*)
  - lehet a kivétel őstípusa is
- Metódus dobásakor megszakad a metódus végrehajtása
- A dobott kivételt a hívási láncban első, a dobottal kompatibilis típust váró *catch* ág kapja el

Kivételek hierarchiája

- Catch olyan, mint egy metódus fejléc
  - kivétel típusa alapján választódik ki
- Lehet, hogy többfajta hiba is történhet
  - kivétel tartalmát nem vizsgáljuk
  - típusa alapján kell dönteni
- Kivételeket öröklési hierarchiába szervezhetjük
  - egységesen tudunk kezelni különféle kivételeket

```
public void foo() throws Exception1, Exception2 { ... }
```

```
try { foo(); }
catch (Exception1 e) { e.printStackTrace(); xxx(); }
catch (Exception2 e) { e.printStackTrace(); yyy(); }
```

Kivételtípusok megadása

- Példakódban egyes hiba-esetek specializálhatók
  - *CourseFullException* extends *Exception*
  - *NeptunNotFoundException* extends *Exception*
- Akár közös, speciális ősük is lehet:
  - *CourseException* extends *Exception*
  - *CourseFullException* extends *CourseException*
  - *NeptunNotFoundException* extends *CourseException*

Kurzus metódusai újra

```
...
public void addStudent(Student s)
throws CourseFullException {
  for (int i = 0; i < students.length; i++) {
    if (students[i] == null) {
      students[i] = s;
      return;
    }
  }
  throw new CourseFullException();
}

public void removeByNeptun(String neptun)
throws NeptunNotFoundException {
...
```

Kurzus metódusai meghívva

```
...
try {
        course.removeByNeptun("EZVANE");
} catch (NeptunNotFoundException e) {
        e.printStackTrace();
}
...
```

Alapkivételek

- *Throwable*
  - minden kivétel őse
  - ez az első dobható osztály a hierarchiában
- *Exception* extends Throwable
  - minden elkapandó kivétel őse
  - kötelező elkapni vagy jelezni

Speciális kivételosztályok

- *RuntimeException* extends Exception
  - nem kötelező elkapni
  - "zajt" vinne a kódba
  - pl. *NullPointerException*, *ArrayIndexOutOfBoundsException*, stb.
- *Error* extends Throwable
  - rendszerhiba
  - nincs értelme elkapni
  - pl. *LinkageError*, *OutOfMemoryError*, stb.

Mit adnak nekünk a kivételek?

- String message
  - leírás a kivétellel kapcsolatban
  - konstruktorban állítható
    - pl. saját kivételben a *super(message)* hívással
  - getMessage() metódussal kérdezhető
- Stacktrace
  - azon metódusok listája, amin a kivétel végigment az eldobástól az elkapásig
  - *printStackTrace()* és barátai
- InitCause
  - egymásba skatulyázott kivételek esetén

StackTrace kibontva

```
52 void foo() throws MyException {
53   if (nagyBaj) throw new MyException("Nagy a Baj!!!");
54 }
55 void bar() throws MyException {
56   // ...
57   foo();
58 }
59 void qux() {
60   try {
61     // ...
62     bar();
63     // ...
64   } catch (MyException me) {
65     me.printStackTrace();
66   }
67 }
```

```
// kimenet:
MyException: Nagy a Baj!!!
        at Main.foo(Main.java:53)
        at Main.bar(Main.java:57)
        at Main.qux(Main.java:62)
        at Main.main(Main.java:96)
```