Objektumorientált programozás

Polimorfizmus és heterogén kollekciók

Goldschmidt Balázs

balage@iit.bme.hu

Statikus tagok

Statikus és példányszintű tagok

- Osztályok és objektumok
  - osztály a típus (*Complex*)
  - objektum az érték, példány (*4+2i*)
- Attribútumok és metódusok
  - tipikusan objektumon működnek (*getRe()*)
- Osztályszintű (static) mezők
  - osztályhoz kötődő elemek
    - tipikusan metódusok
  - lehetnek attribútumok is
    - elérhetők a példányokból is
  - minden példány számára közösek

Statikus és példányszintű tagok

- Complex kiíratása *i* vagy *j*?

```
public class Complex {
    private double re, im;
    public Complex(double r, double i) {
        re = r; im = i;
    }
    double getRe() { return re; }
    //...
    private static char e = 'i';
    public String toString() {
        return re + ((im < 0)? "" : "+") +im + e;
    }
    static public setE(char c) { e = c; }
}
```

statikus adattag

Statikus és példányszintű tagok

- Complex kiíratása *i* vagy *j*?

```
Complex c1 = new Complex(3,4);
Complex c2 = new Complex(1,-5);

System.out.println(c1); // 3+4i
System.out.println(c2); // 1-5i

Complex.setE('j');

System.out.println(c1); // 3+4j
System.out.println(c2); // 1-5j
```

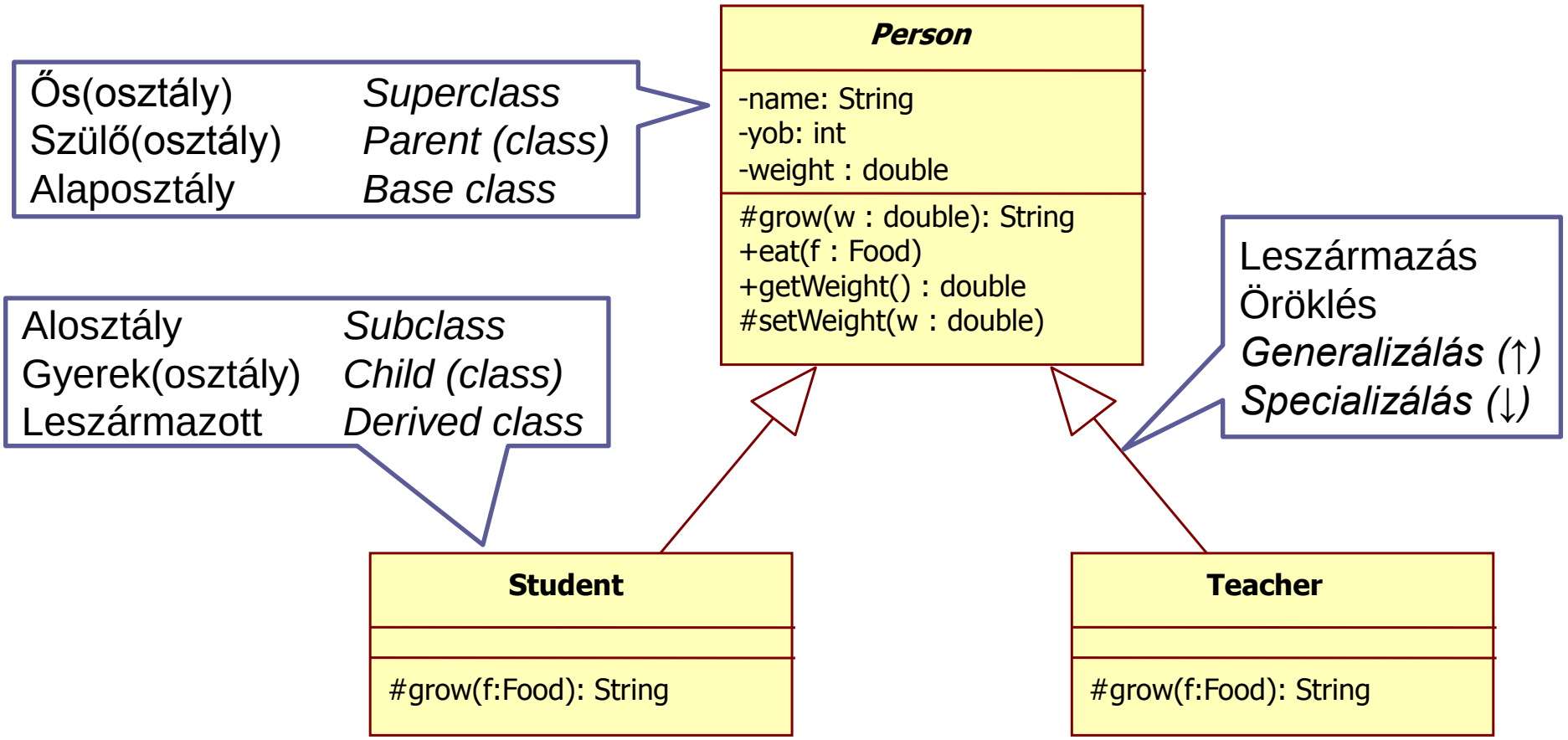
Tipikusan osztályon hívjuk

Ismétlés

Ismétlés: az 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-elv, konform öröklés, kompatibilitás

Elnevezések



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 dinamikus típus metódusa fut (ha van)

```
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 = 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

Virtuális metódusok újra

Virtuális metódusok, példa 1

- Lehessen enni és hízni!

```
public class Person {
  // ...
  private int weight; // testsúly, privát
  public double getWeight() { return weight; }
  protected void setWeight(double w) { //setter, prot.
    weight = w;
  }
  protected void grow(double w) { // ennyivel nő a súlya
    weight +=  w;
  }
  public void eat(Food f) {
    this.grow(f.getNutrients());
  }
...
```

Virtuális metódusok, példa 2

- Hallgató csak félig hízik!
  - módosítsuk a grow metódust!

```
public class Student extends Person {
  // ...
  protected void grow(double w) {
    this.setWeight(this.getWeight()+w/2);
  }
...
```

Elhagyható

```
Student s = new Student("Gipsz Jakab", "1A2B3C", 1996);
Food f = new Food(2.4); // 2.4kg pizza
s.eat(f);
```

Mi fog történni?

Virtuális metódusok, példa 3

- Kinek a metódusa fut?

```
Student s = new Student("Gipsz Jakab", "1A2B3C", 1996);
Food f = new Food(2.4); // 2.4kg pizza
s.eat(f); // Person.eat
  →this.grow(w=f.getNutrients())
    →this.setWeight(this.getWeight()+w/2)
```

this statikus típusa *Person*, din. típusa *Student* →*Student.grow*

s statikus és din. típusa *Student*

this statikus típusa *Person*, din. típusa *Student* →*Student.setWeight* lenne

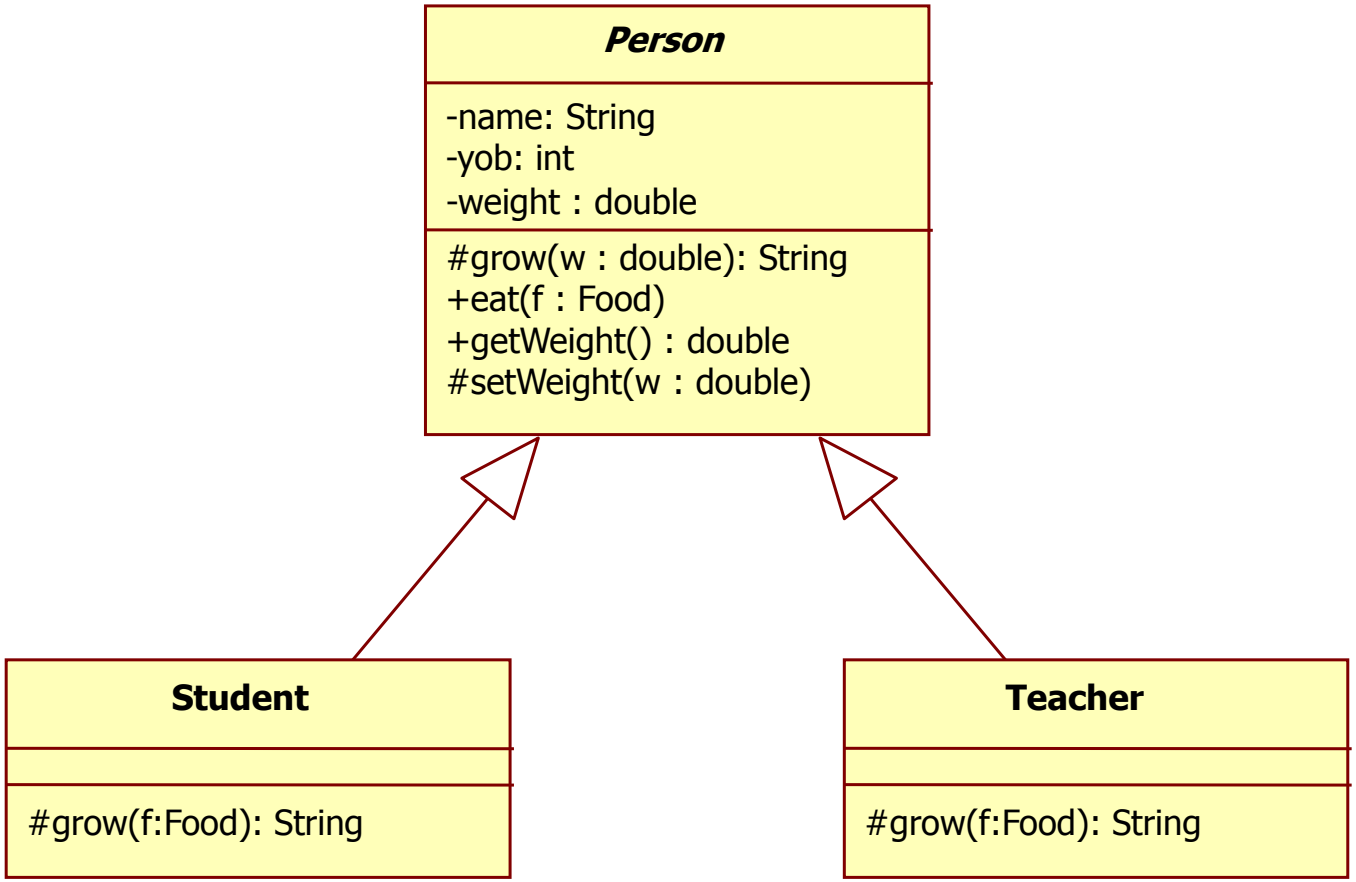
Virtuális metódusok, példa 4

```
public class Person {
  // ...
  private int weight; // testsúly, privát
  public double getWeight() { return weight; }
  protected void setWeight(double w) { this.weight = w; }
  protected void grow(double w) { this.weight +=  w; }
  public void eat(Food f) { this.grow(f.getNutrients());}
...
```

```
public class Student extends Person {
  // ...
  protected void grow(double w) {
    this.setWeight(this.getWeight()+w/2);
  }
...
```

Polimorf viselkedés

UML jelölés


Person
-name: String
-yob: int
-weight : double
#grow(w : double): String
+eat(f : Food)
+getWeight() : double
#setWeight(w : double)
Student
#grow(f:Food): String
Teacher
#grow(f:Food): String

Virtuális metódusok, final

```
public class Person {
  // ...
  private int weight; // testsúly, privát
  public double getWeight() { return weight; }
  final protected void setWeight(double w) {
    this.weight = w;
  }
  protected void grow(double w) { this.weig  += w; }
  public void eat(Food f) { this.grow(f.getNu   ts()); }
...
```

Nem felüldefiniálható

- *final* módosító
  - metódus: nem lehet felüldefiniálni
  - attribútum, változó: konstans érték, egyszer lehet inicializálni

Öröklés szerepe

- Bottom-up: Kiemelni a közös jellemzőket
  - azonos tartalmú attribútum, metódus
  - csökken a kódméret
  - csökken a többszörözés
  - csökken a csatolás mértéke
- Top-down: bővíteni a meglevő osztályok felelősségét
  - új attribútum vagy metódus
  - meglevő metódus felüldefiniálása

DRY: don't repeat yourself

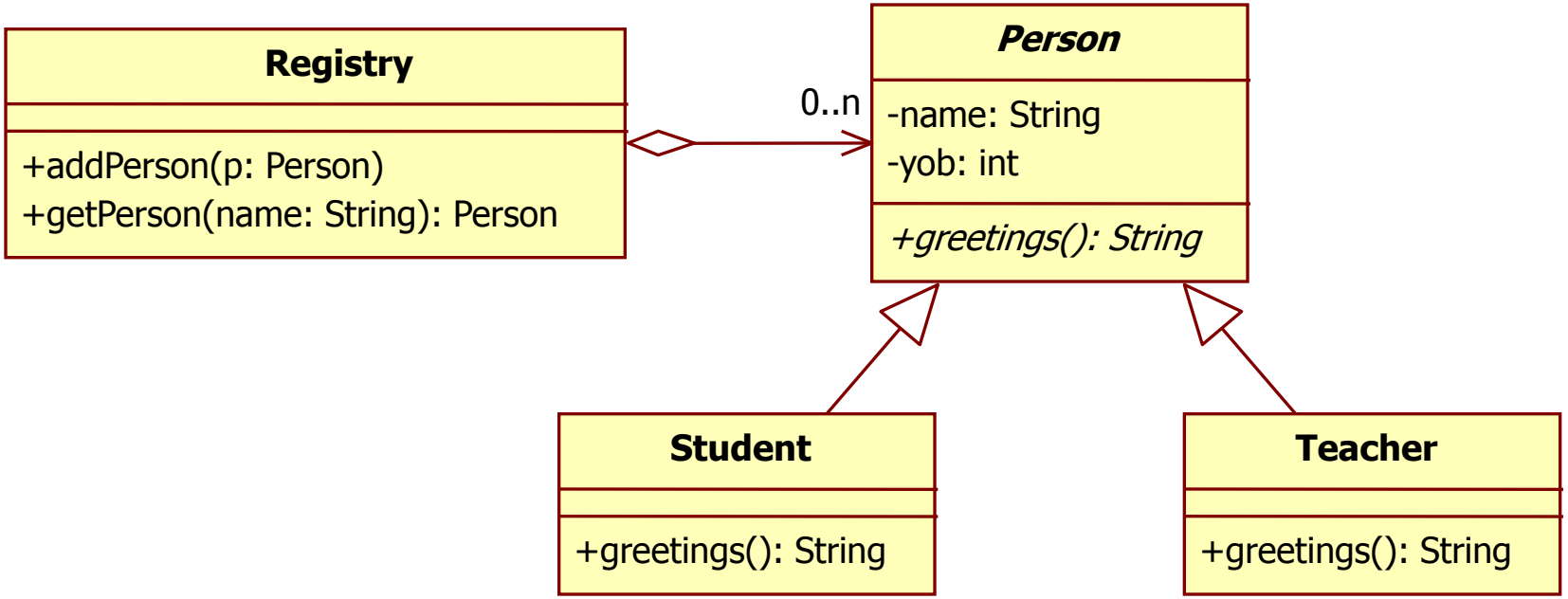
OCP: open-closed principle

OO elvek

- DRY: don't repeat yourself
  - ismétlődés hibához vezet, kerüljük
  - metódusba emelés
  - ősbe emelés
- OCP: open-closed principle
  - nyitott a bővítésre
  - zárt a változtatásra
    - új leszármazott
    - új metódus

Heterogén kollekció

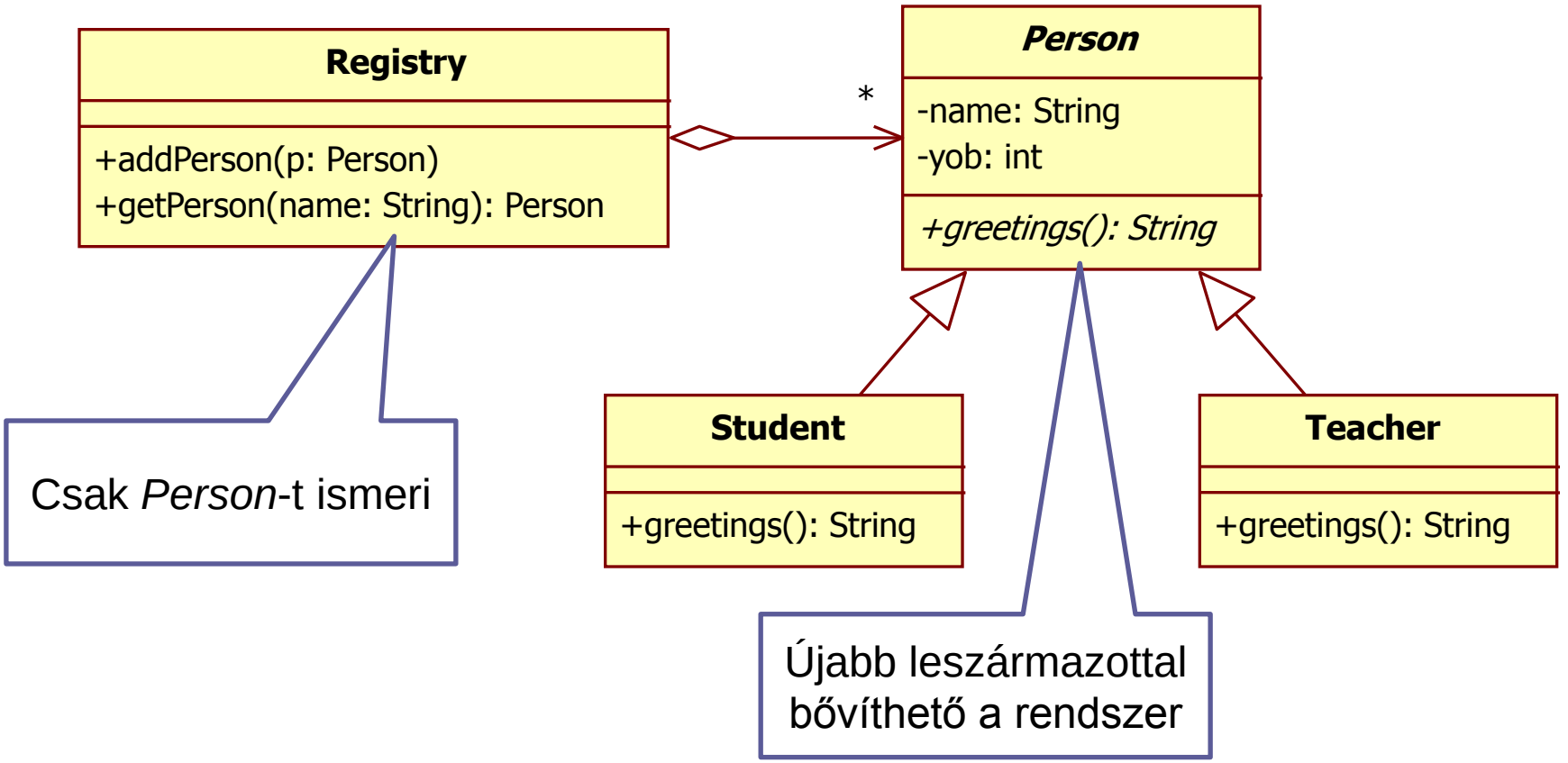
Példa: Registry tároljon



Polimorfizmus

- Egy objektum azon tulajdonsága, hogy sokfajta formát felvehet
  - OO-ban: ugyanaz a referencia különböző típusú objektumokra mutathat anélkül, hogy tudnunk kellene, mi a pontos típus
- Az öröklés teszi lehetővé
  - statikus típus mutat és definiálja a formát (felületet)
  - dinamikus típus hajt végre, definiálja a tartalmat (működést)

Példa


Registry
+addPerson(p: Person)
+getPerson(name: String): Person
*
Person
-name: String
-yob: int
+greetings(): String
Student
+greetings(): String
Teacher
+greetings(): String
Csak Person-t ismeri
Újabb leszármazottal bővíthető a rendszer

Registry implementációja (tömb)

```
public class Registry {
  private Person persons[];
  public Registry(int n) { // max n Person-t tárol
    persons = new Persons[n]; // null-okkal tele
  }
  public void addPerson(Person p) {
    for (int i = 0; i < persons.length; i++) {
      if (persons[i] == null) { persons[i] = p; return; }
    }
  }
  public Person getPerson(String n) { // ha nincs, null
    for (int i = 0; i < persons.length; i++) {
      if (persons[i].getName.equals(n)) return persons[i];
    }
    return null;
  }
}
```

Registry implementációja (lista)

```
public class Registry {
  private ArrayList<Person> persons;
  public Registry(int n) { // max n Person-t tárol
    persons = new ArrayList<Person>(); // üres lista
  }
  public void addPerson(Person p) {
    persons.add(p); // delegálunk a listához
  }

  public Person getPerson(String n) { // ha nincs, null
    for (int i = 0; i < persons.size(); i++) {
      if (persons.get(i).getName.equals(n))
        return persons[i];
    }
    return null;
  }
}
```

Heterogén kollekció

- Polimorf objektumok gyűjteménye
  - *Registry* nem tudja, kik a *Person*-ök
  - keveredhetnek a diákok és az oktatók
  - csak a kollekció statikus osztálya (*Person*) ismert

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Teacher t1 = new Teacher("Rend Elek", "Q1W2E3", 1973);

Registry r = new Registry(10);
r.addPerson(s1);
r.addPerson(t1);
Person p = r.getPerson("Rend Elek");
System.out.println(p.greetings());
```

Kohézió és csatolás

- Kohézió
  - osztályon belüli függések mértéke
    - minél nagyobb, annál jobb: a felelősség tömörül
    - ha alacsony, "szétesik" az osztály: érdemes megbontani
- Csatolás
  - két osztály vagy objektum közötti függés mértéke
  - minél kisebb, annál jobb
    - későbbi változások nem terjednek messzire

Heterogén kollekció előnyei

- Csökkenti a csatolást
  - a kollektor csak az őst ismeri

  → további leszármazottak felvétele nem befolyásolja

  → a kollekcióban tárolt típusok száma szabadon nőhet
- Növelheti a kohéziót
  - a leszármazottak csak a rájuk specifikus dolgokért felelősek

Bővítsük a kollekciót

- Legyen új *Person* típus:
  - Kutató (*Researcher*)
  - Köszönés: "Szervusz, kérlek!"
  - Extra felelősség
    - kutatási terület (*topic*), getter-setter-rel

Bővítsük a kollekciót

```
public class Researcher extends Person {
  private String topic;

  public Researcher(String na, String ne, int y) {
    super(na,ne,y);
    title = "assistant teacher";
  }
  public void setTopic(String s) {
    topic = s;
  }
  public String toString() {
    return super.toString() +", "+topic;
  }
  public String greetings() {
    return "Szervusz, kérlek!";
  }
}
```

Új osztály használata

```
Student s1 = new Student("Gipsz Jakab", "1A2B3C", 1996);
Teacher t1 = new Teacher("Rend Elek", "Q1W2E3", 1973);
Researcher r1 =
  new Researcher("Kuta Tóbiás", "314159", 1983);

Registry r = new Registry(10);
r.addPerson(s1);
r.addPerson(t1);
r.addPerson(r1);
Person p = r.getPerson("Kuta Tóbiás");
System.out.println(p.greetings());
```

Registry működik, nem kellett módosítani

Heterogén kollekció szabálya

- Sose használjunk típusinformációt
  - ne kódoljuk a leszármazott típust
    - getType, getTypeID, instanceOf, stb.
  - elrontja a heterogenitást
  - csatolást növeli
  - felelősséget a használóba helyezi
    - a Person leszármazottja tudja, mit kell csinálni
    - a Person használója ne komparáljon típusra, mert mi lesz, ha jön egy új típus?

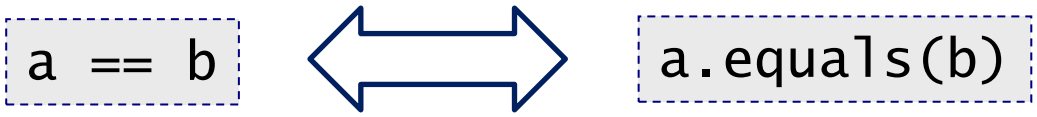
Object osztály

Minden polimorfizmus őse

- *Object* osztály minden osztály őse
  - definiál alapmetódusokat
  - így minden objektumon meg lehet hívni
  - az egységes működés garantált
- Metódusai
  - `public String toString()`
  - `public boolean equals(Object o)`
  - *`protected Object clone()`*
  - *`public int hashCode()`*
  - `...`

Objektumok azonossága

- `==` operátor
  - referencia-alapú azonosság
    - ugyanaz-e a két objektum?
- `boolean equals(Object o)`
  - tartalom-alapú azonosság
    - ugyanaz-e a tartalmuk?
  - rekurzió javasolt
    - ha a tartalom is equals, akkor az objektumok is
  - *alapértelmezett megvalósítás referencia-alapú*



Person equals

- Neptunkód azonossága

```
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 boolean equals(Object o) {
        Person p = (Person)o;
        return (neptun.equals(p.neptun));
    }
}
```

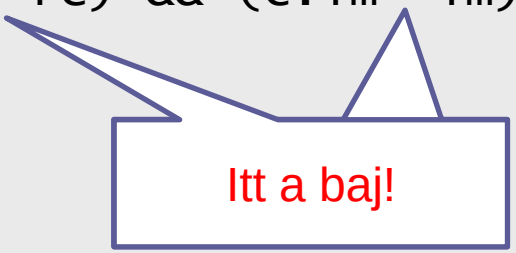
Kasztolni kell

Complex equals (rossz)

```
public class Complex {
    private double re, im;
    public Complex(double r, double i) {
        re = r; im = i;
    }
    double getRe() { return re; }
    //...

    public boolean equals(Object o) {
        Complex c = (Complex)o;
        return (c.re==re) && (c.im==im);
    }


}
```



Complex equals (jó)

```
public class Complex {
    private double re, im;
    public Complex(double r, double i) {
        re = r; im = i;
    }
    double getRe() { return re; }
    //...
    private static double delta = 1e-6; // hibahatár
    private static boolean close(double a, double b) {
        return Math.abs(a-b) < delta;
    }
    public boolean equals(Object o) {
        Complex c = (Complex)o;
        return (close(re, c.re) && close(c.im, im));
    }
}
```