Java VI.
Öröklődés

Ficsor Lajos

Miskolci Egyetem

Általános Informatikai Tanszék

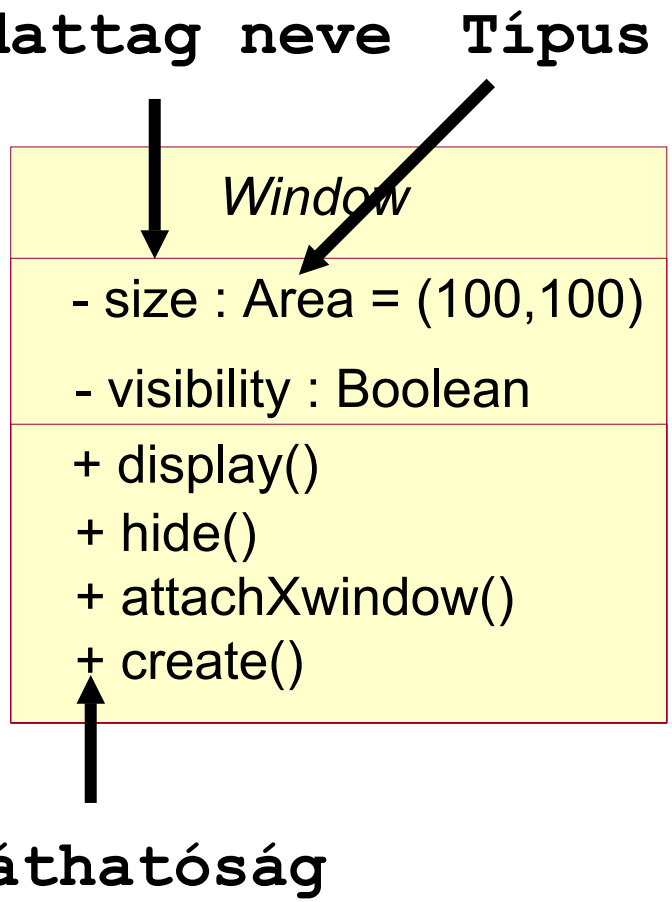
Utolsó módosítás: 2006. 03. 07.

Egy kis kitérő: az UML

- UML: Unified Modelling Language
- Grafikus eszköz objektum orientált modellek készítésére
- Osztálydiagram: osztályok és kapcsolataik ábrázolására
- Az UML további diagramm típusokat is definiál

Osztály diagram

Osztály szimbóluma



Osztály neve

Adattagok

Metódusok

A láthatóság jelölése:

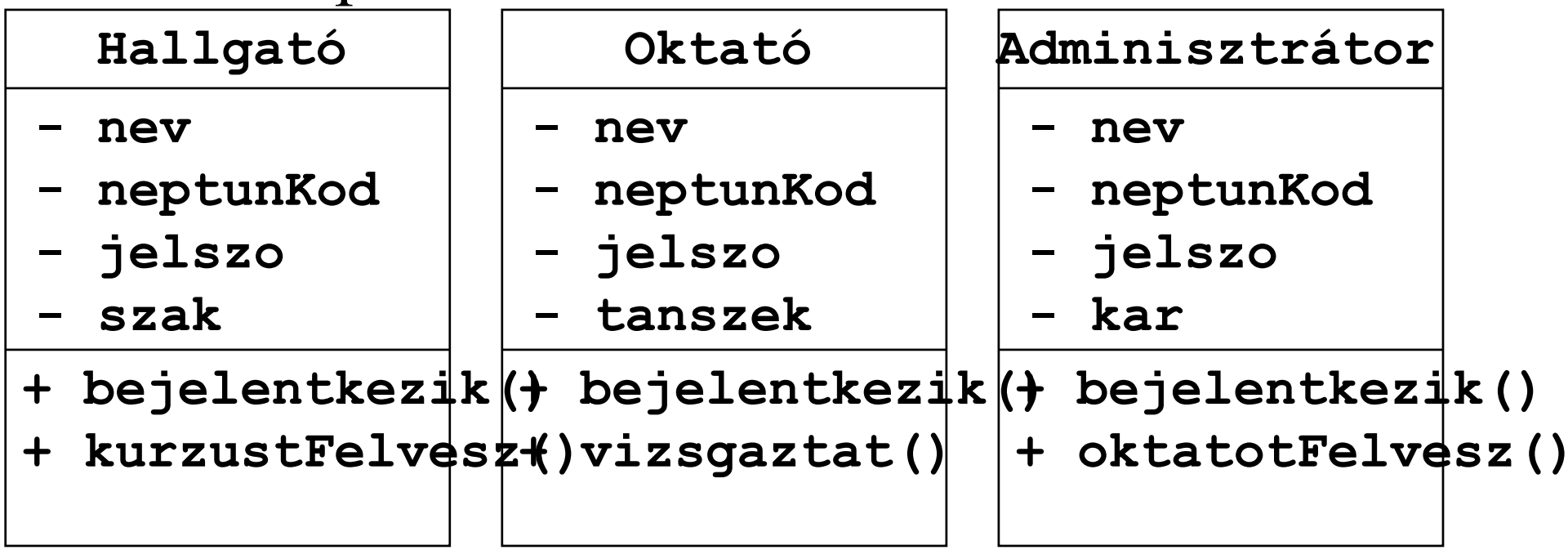
\+ public

\# protected

\- private

Osztályok közös jellemzőkkel

- Gyakran előfordul, hogy egy program különböző osztályai azonos adattagokkal és metódusokkal rendelkeznek
- Példa: Neptun felhasználók



Osztályok közös jellemzőkkel (folyt.)

- Az előző példában minden osztályban szerepel három azonos adat
  - **`nev`**
  - **`neptunKod`**
  - **`jelszo`**
- Minden osztályban szerepel a **`bejelentkezik()`** metódus
- Az osztályok ezen kívül rendelkeznek csak rájuk jellemző adattagokkal és metódusokkal is.

Osztályok közös jellemzőkkel (folyt.)

- Az adattagok és metódusok ismétlődése problémákat okozhat:
  - pazarlás
  - metódusoknál biztosítani kell az azonos működést
  - a módosításokat több helyen át kell vezetni.
- A megoldás: **öröklődés**

Az öröklődés fogalma

- Egy osztály deklarálható valamely más osztály (ős osztály vagy szülő osztály) leszármazottjaként.
- A leszármazott osztály rendelkezik
  - a szülő osztály tagjaival
  - a saját tagjaival.
- Az ős osztály elemeinek az elérése a leszármazott osztályból nem feltétlenül garantált.
- Az öröklődési hierarchia tetszőleges lehet.
  - Egyetlen korlátozás: egy osztály még közvetett módon sem lehet saját maga őse.

Az öröklődés fogalma (folyt.)

- Az ős osztály továbbra is használható önmagában is.
- Ha egy Java osztálynak nincs megadva őse, automatikusan az **`Object`** osztály leszármazottja lesz. Minden osztálynak van tehát egy közös őse.
- Angol kifejezések
  - öröklődés: inheritance
  - ősosztály: base class
  - leszármazott osztály: derived class

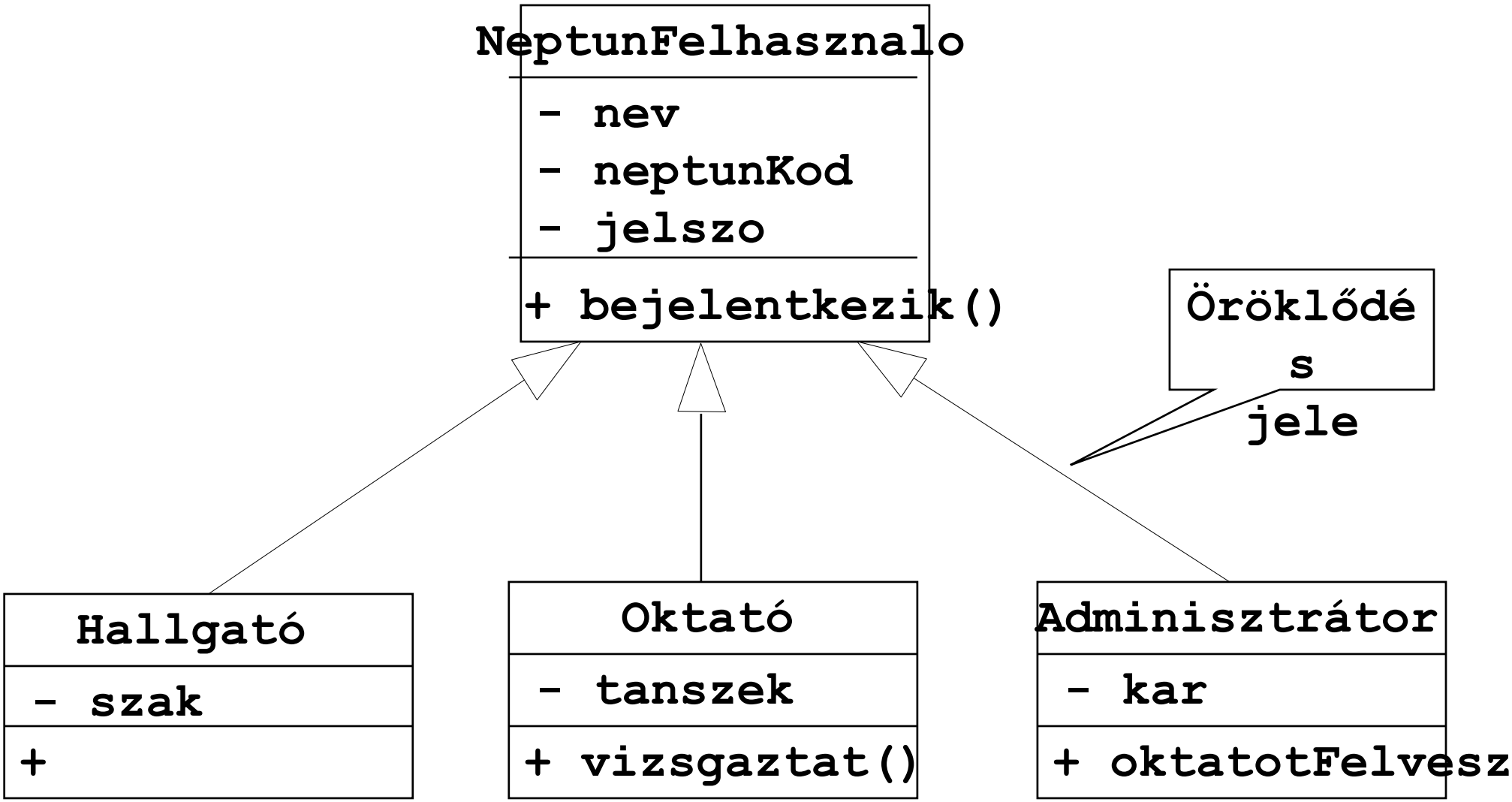
Szintaktika

```
[módosító] class név extends ősosztály
{

// itt jön a saját tagok
// deklarációja

}
```

A példa örökléssel

- A közös adatok és metódusok egy ős osztályba gyűjthetők.
- Ennek neve legyen **`NeptunFelhasznalo`**
- Az egyes felhasználó osztályok ezek leszármazottjaiként deklarálhatók.
- A leszármazottak öröklik a közös tagokat.
- Ezzel kód megosztás jön létre: a leszármazottak használhatják a szülő osztály metódusait.

A példa örökléssel (folyt.)



Hivatkozás a leszármazottra

- Mivel egy leszármazott az őse minden tulajdonságával rendelkezik, bármikor használható ős típusú objektumként is. (Fordítva általában nem igaz!)
- Ezért egy ős típusú hivatkozás használható leszármazott típusú objektumhoz is.
  - Következmény: bármely objektumra hivatkozhatunk `Object` típusú hivatkozással.
- Egy változónak van **statikus** és **dinamikus** típusa.

Statikus és dinamikus típus

- Egy változó statikus típusa az, amelyet a deklarációjában megadtunk.
  - Ez a változó teljes élete alatt változatlan.
- Egy változó dinamikus típusa az általa éppen hivatkozott objektum tényleges típusa.
  - Ez a program futása során bármikor változhat.
- A változó dinamikus típusa csak a statikus típus vagy annak leszármazottja lehet.
- A későbbikben ennek fontos szerepe lesz!

Hozzáférés a leszármazottból

- A leszármazott osztály az ős osztályból örökölt tagokra hozzáférés szempontjából ugyanolyan jogokkal rendelkezik, mint bármely más osztály.
  - Például az örökölt **`private`** adattagot nem érheti el közvetlenül.
  - Mivel azonban az örökölt adattagok a részét képezik, az örökölt **`public`** metóduson keresztül használhatják.
- A leszármazottra vonatkozó speciális minősítő a **`protected`**.

A `protected` hozzáférési kategória

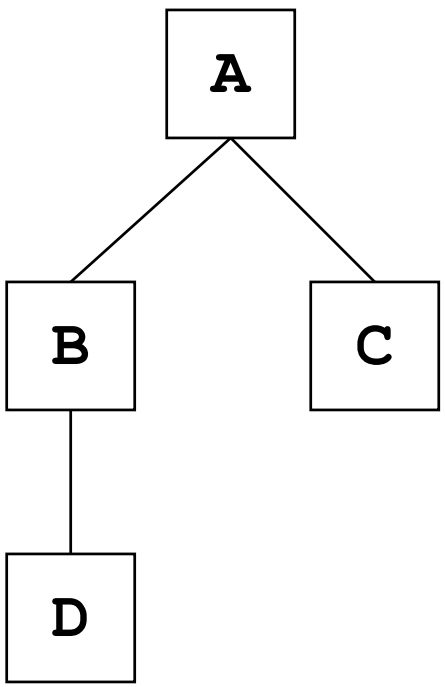
A félnyilvános kategória kiterjesztése

A **`protected`** minősítésű taghoz hozzáférhetnek:

- az azonos csomagban levő osztályok,
- egy másik csomagban definiált leszármazott osztály, ha
  - minősítés nélkül hivatkozik rá vagy
  - saját, vagy leszármazottja típusának megfelelő minősítéssel hivatkozik rá.

A `protected` hozzáférés (példa)

- Legyen az alábbi osztály hierarchia, minden osztály külön csomagban:



  - Ha **`A`**-nak van egy **`p`** protected tagja, **`B`** hozzáférhet
    - csak a nevével
    - **`B`** vagy **`D`** típusú minősítéssel.
  - Nem férhet hozzá **`C`** típusú minősítéssel

Egyszerű példa: jarmu osztály

```
package jarmu1;
public class jarmu {
  private int kerekek;
  private double suly;
  public void kezdoertek(int k, double s)
     {kerekek = k; suly = s;}
  public int kerekszam (){ return kerekek;}
  public double kerekterheles()
     {return suly / kerekek; };
  public double sulya()  {return suly;}
}
```

Egyszerű példa: gepkocsi osztály

```
package jarmu1;


public class gepkocsi extends jarmu {
  private int szemelyek;
  public void kezdoertek(int k, double s,
  int szem)
    {szemelyek = szem;
    kezdoertek(k, s); }     // Ez a jarmu
                            // osztalyból
}                           // örökölt
```

Egyszerű példa: teherauto osztály

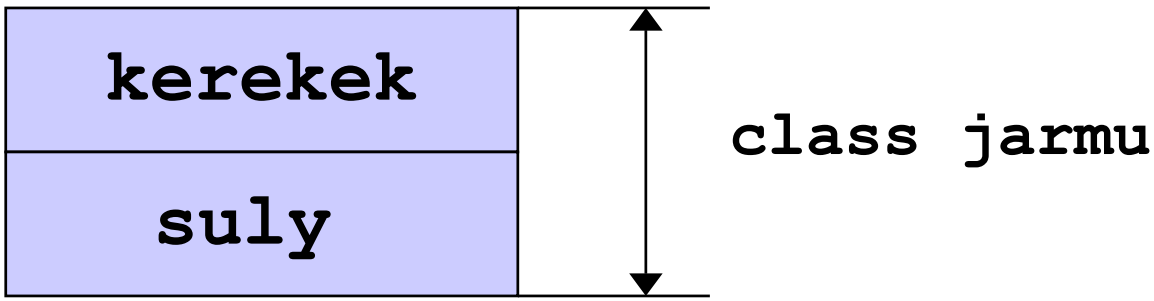
```
package jarmu1;
public class teherauto extends jarmu {
   private int szemelyek;
   private double raksuly;
   public void teher_kezdoertek
      (int szem,double rs)
      {szemelyek = szem; raksuly = rs; }
   public double kerekterheles()
      {return (sulya() + raksuly +
       szemelyek*60.) / kerekszam(); }
// suly + … hibas lenne, mert suly
// nem elérhető!!!
}
```

Egyszerű példa: `bicikli` objektum

```
package jarmu1;
public class proba {
  public static void main(String[] args) {
   jarmu bicikli = new jarmu();
   bicikli.kezdoertek(2,15.0);
   System.out.print("Bicikli kerekeinek szama:");
   System.out.println(bicikli.kerekszam());
   System.out.print("Bicikli kerekterhelese: ");
   System.out.println(bicikli.kerekterheles());
```

A `bicikli` objektum felépítése

- Az objektum típusa **`jarmu`**

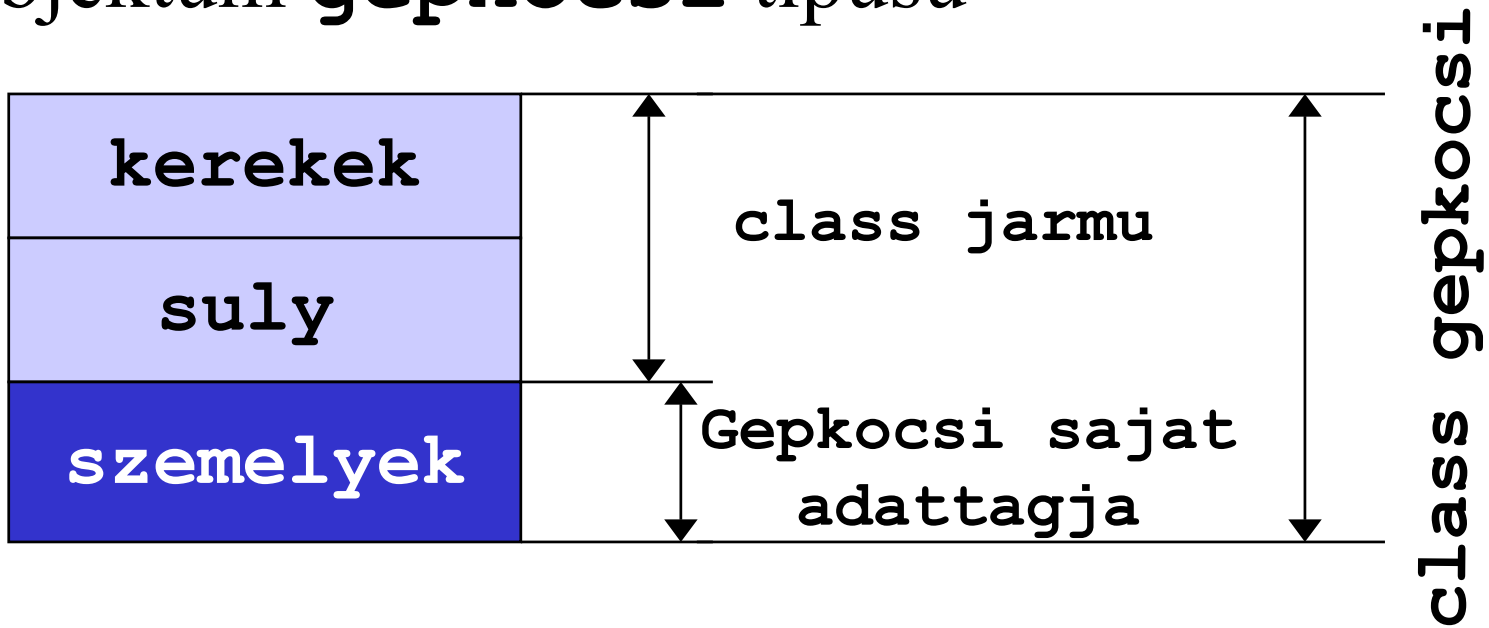


Egyszerű példa: trabant objektum

```
gepkocsi trabant = new gepkocsi();
trabant.kezdoertek (4, 600.0, 4);
System.out.print("Trabant kerekeinek szama: ");
System.out.println(trabant.kerekszam());
```

A `trabant` objektum felépítése

- Az objektum **`gepkocsi`** típusú



Egyszerű példa: `ifa` objektum

```
teherauto ifa = new teherauto();
ifa.kezdoertek(6, 1200.0);
ifa.teher_kezdoertek(2,2500.0);
System.out.print("Max. kerekterheles: ");
System.out.println(ifa.kerekterheles());



} // main() vege
} // class proba vege
```

Egyszerű példa: eredmény

```
// Futas eredmenye:
//   Bicikli kerekeinek szama: 2
//   Bicikli kerekterhelese: 7.5
//   Trabant kerekeinek szama: 4
//   Max. kerekterheles: 636.6666666666666
```

Egyszerű példa: megjegyzések

- A leszármazott osztály definiálhat ugyanolyan nevű példánymetódust, mint amit örökölt.

Az eredmény lehet:

- **Metódusnév túlterhelés**, ha a paraméter szignatúra különböző.
  - Ilyenkor egy hatáskörben van az örökölt és a saját változat.
- **Metódus felüldefiniálás**, amelyre további szabályok vonatkoznak (lásd később).
- A példában metódusnév túlterhelést alkalmaztunk.

Egyszerű példa: megjegyzések (folyt.)

- A **`jarmu`** osztály tartalmaz **`suly`** adattagot.
- Az adatag **`private`**, tehát nem lehet rá közvetlenül hivatkozni az osztályon kívül. (A leszármazottban sem!)
- A **`sulya()`** metódus **`public`**, ezt használhatja a **`teherauto`** osztály.
- Ezzel közvetve az örökölt **`suly`** adattag saját példányát éri el.

Egyszerű példa: megjegyzések (folyt.)

- Az osztályok csak implicit konstruktort tartalmaznak. (Nem definiáltunk konstruktort.)
- A **`main`** metódusban használt mindhárom változó statikus és dinamikus típusa megegyezik.

Egyszerű példa: megjegyzések (folyt.)

A mintapélda számos **tervezési hibával** terhelt!

- Konstruktorok helyett kezdőérték függvény - lehetővé teszi rosszul inicializált objektumok létrehozását.
- A **`gepkocsi`** osztály örökli a **`kerekterheles`** függvényt, amely azonban hibás eredményt ad, mert nem veszi figyelembe a személyek súlyát.
- A **`teherautó`** osztály inicializálása csak két metódus hívással teljes, ami lehetővé teszi részlegesen inicializált objektumok létrehozását.

Konstruktorok öröklődés esetén

- A konstruktor nem öröklődik (nem metódus).
- Mind az ős osztály, mind a leszármazott osztály rendelkezhet konstruktorral (akár többel is).
- Egy leszármazott objektum példányosításánál tisztázni kell:
  - a konstruktorok végrehajtási sorrendjét,
  - azt, hogy hogyan választhatjuk ki az ősosztály konstruktorai közül a végrehajtandót.

Konstruktorok végrehajtási sorrendje

- Először mindig az ős osztály, majd a leszármazott osztály konstruktora hajtódik végre.
- A pontos sorrend:
  - az ős osztály adattagjainak inicializálása (az inicializátor kifejezések kiértékelésével),
  - az ős osztály konstruktorának végrehajtódása,
  - a gyermek osztály adattagjainak inicializálása,
  - a gyermek osztály konstruktorának végrehajtódása.

Ős osztály konstruktorának kijelölése

- A gyermek osztály első sorában szerepelhet egy

  `super(paraméterek)`

konstruktorhívás.

- A paraméterlistának az ős osztály valamelyik konstruktorára illeszkednie kell.
- Ha ilyen hívás nem szerepel a gyermek osztály konstruktorában, akkor egy implicit

  `super()`

hívással kezdődik a konstruktor végrehajtása.

Ős osztály konstruktorának kijelölése (2.)

- Következmények:
  - Ha a gyermek osztálynak van olyan konstruktora, amelyben nincs explicit ős konstruktor hívás, a szülő osztálynak kell legyen paraméter nélküli konstruktora.
  - Ha a gyermek osztálynak csak implicit konstruktora van, az is az ős osztály paraméter nélküli konstruktorát hívja meg.
  - A fenti szabályok megsértését a fordító hibajelzéssel honorálja!

Egyszerű példa konstruktorral

```
package jarmu1;
public class jarmu {
  private int kerekek;
  private double suly;
  public jarmu(int k, double s)
     {kerekek = k; suly = s;}
  public int kerekszam (){return kerekek;}
  public double kerekterheles()
     {return suly / kerekek; };
  public double sulya()  {return suly;}
}
```

Egyszerű példa konstruktorral (folyt.)

```
package jarmu1;
public class gepkocsi extends jarmu {
  private int szemelyek;
  public gepkocsi(int k, double s,
    int szem){
      super(k, s);
      szemelyek = szem;
  }
}
```

Egyszerű példa konstruktorral (folyt.)

```
package jarmu1;
public class teherauto extends jarmu {
  private int szemelyek;
  private double raksuly;
  public teherauto (int k, double s,
    int szem,double rs) {
      super(k,s);
      szemelyek = szem; raksuly = rs; }
  public double kerekterheles()
     {return (sulya() + raksuly +
  szemelyek*60.) / kerekszam(); }
}
```

Egyszerű példa konstruktorral (folyt.)

```
package jarmu1;
public class proba {
  public static void main(String[] args) {
  jarmu bicikli = new jarmu(2, 15.0);
  System.out.print("Bicikli kerekeinek szama: ");
  System.out.println(bicikli.kerekszam());
  System.out.print("Bicikli kerekterhelese: ");
  System.out.println(bicikli.kerekterheles());
```

Egyszerű példa konstruktorral (folyt.)

```
gepkocsi trabant = new gepkocsi(4, 600.0, 4);
System.out.print("Trabant kerekeinek szama: ");
System.out.println(trabant.kerekszam());

teherauto ifa = new teherauto(6, 1200.0,
                                    2,2500.0);
System.out.print("Max. kerekterheles: ");
System.out.println(ifa.kerekterheles());

  }
```

Az `Object` osztály

- Ha egy osztálydefinícióban nincs megadva ős osztály, akkor automatikusan az Object osztály lesz az ős.
- Minden osztály automatikusan örököl bizonyos metódusokat.
- Az örökölt metódusok egy része felüldefiniálható.

`final` osztályok

- A **`final`** módosítóval megjelölt osztályoknak nem lehet leszármazottja.
  - Az ilyen osztály működése nem változtatható meg az öröklődési mechanizmussal.
- A módosítót a hozzáférési kategória módosítója után írjuk. Például:

```
public final class System { … }
```