Programozás III

OOP ALAPOK

OBJEKTUM-ORIENTÁLT PARADIGMA

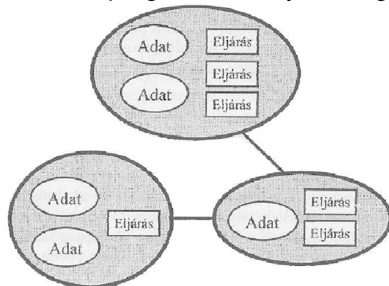
Olyan programozási módszer, amely lehetővé teszi különböző bonyolult változók (objektumok) létrehozását és kezelését.

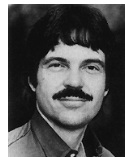
Objektum-orientált program: Egymással kapcsolatot tartó, együttműködő objektumok összessége, ahol minden objektumnak megvan a jól meghatározott feladata.

Az objektumorientált modellezés során elvonatkoztatunk, megkülönböztetünk, osztályozunk, általánosítunk, leszűkítünk, kapcsolatokat építünk, stb. ⇒ absztrahálunk.

OBJEKTUM-ORIENTÁLT PARADIGMA

Alan Kay diplomamunkája 1969: Tervek és elképzelések a SmallTalk programozási nyelv megtervezésére.





The best way to predict the future is to invent it. (1971)

OBJEKTUM-ORIENTÁLT GONDOLKOZÁS, MODELLEZÉS

Alapja: az emberi gondolkozás, hiszen

Megfigyeljük a világot, és közben

– osztályozzuk a látottakat:

a hasonló dolgok egy osztályba kerülnek;
a különbözőek más-más osztályba

így alakulnak ki a fogalmaink

Eközben a dolgokat megkülönböztetjük a számunkra lényeges tulajdonságaik, viselkedési módjuk alapján.

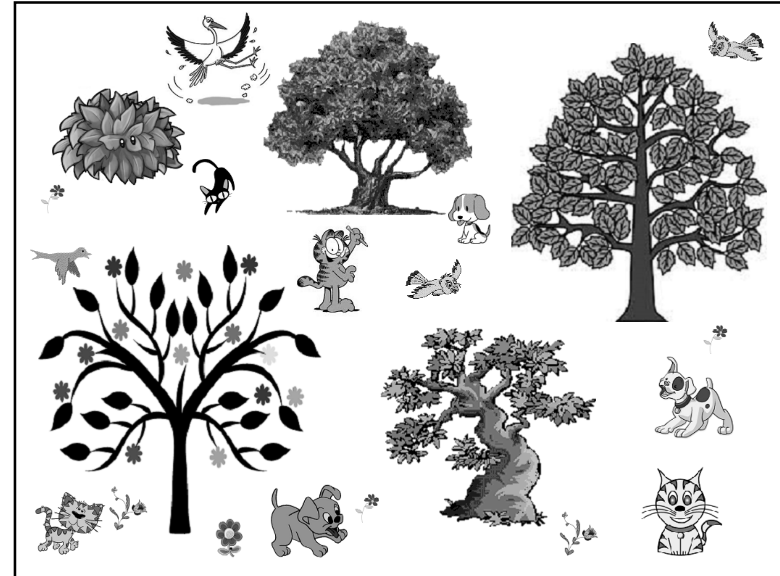
OBJEKTUM-ORIENTÁLT GONDOLKOZÁS, MODELLEZÉS

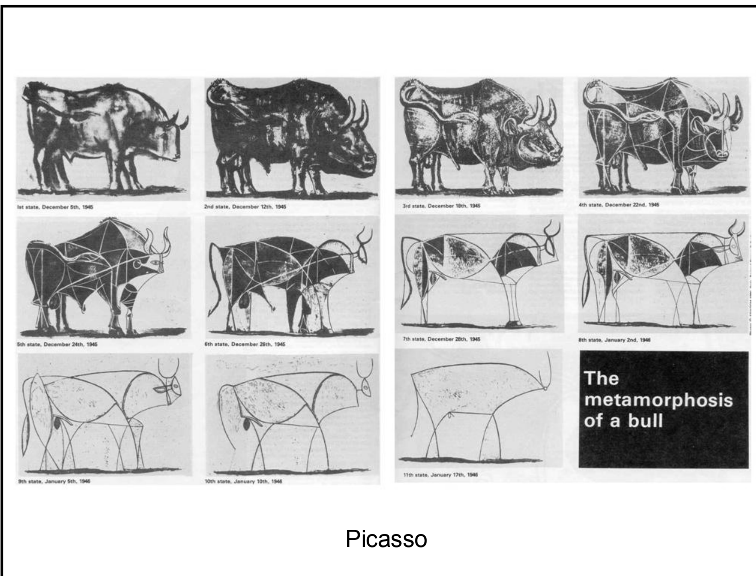
A világ megfigyelése közben

– absztrahálunk:

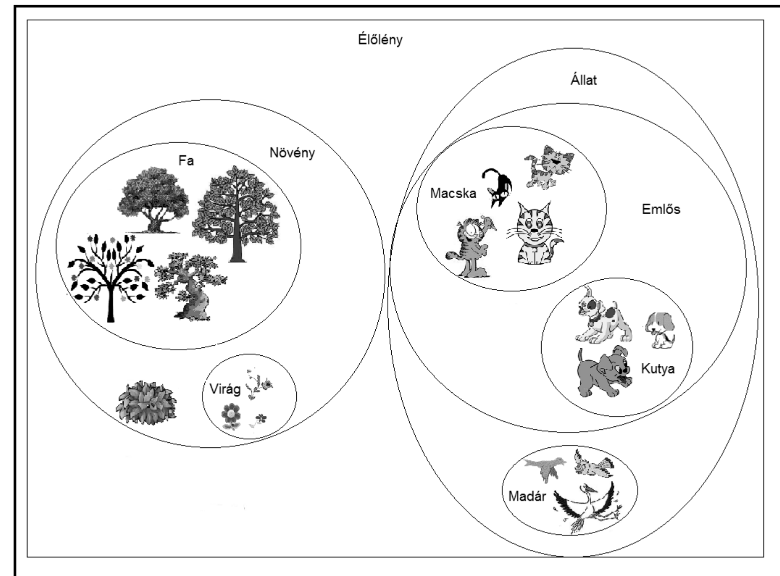
Elvonatkoztatunk a számunkra pillanatnyilag nem fontos, közömbös információktól, és kiemeljük az elengedhetetlen fontosságú részleteket.

Azaz leegyszerűsítjük a valós világot úgy, hogy csak a lényegre, a cél elérése érdekében feltétlenül szükséges részekre összpontosítunk.





Picasso



Most egy közös beszélgetés a témáról,

majd önálló feldolgozás

OBJEKTUM-ORIENTÁLT PARADIGMA – PÉLDA

Melyik a jó metódus?

```
public int eladasiAr(){
    return (int) (this.mennyiseg * this.egysegAr *(1 + Ital.afa / 100.));
}
```

vagy:

```
public int eladasiAr(double mennyiseg, int egsegAr){
    return (int) (this.mennyiseg * this.egysegAr *(1 + Ital.afa / 100.));
}
```

vagy:

```
public int eladasiAr(double mennyiseg, int egysegar){
    return (int)(mennyiseg * egysegar * (1 + Ital.afa / 100.));
}
```

Mikor kell pont a 100 után?

OOP ALAPFOGALMAK

osztály - objektum

Az Osztaly típusú változó majd Osztaly típusú objektumot tartalmaz.

példány, példányváltozó, példánymetódus,
osztályváltozó, osztálymetódus

Példány létrehozása, megszüntetése

Nincs destruktor, helyette szemétgyűjtés
(garbage collection)

OBJEKTUM-ORIENTÁLT PARADIGMA – PÉLDA

1. Miért nem jó?

```
public void termekAr(){
    eladasiAr= afa*literenkentiEgysegAr;
}
```

2. Miért nem jó, ha mindent a beolvasás metódusban oldunk meg? (összegzést, max-keresést, stb.)

3. Miért NINCS kiíratás az alaposztályokban?

KITÉRŐ – MÁSIK PÉLDA

Melyik a jó metódus:

```
public double terulet(){
    return hosszusag*szelesseg;
}
```

vagy:

```
public double terulet(double hosszusag, double szelesseg){
    return hosszusag*szelesseg;
}
```

Elvileg lehet-e jó a második változat?

Hogyan kellene kijavítani, hogy tényleg jó is legyen?

public **static** double terulet(…)

KITÉRŐ: DÁTUMKEZELÉS

```
import java.util.Calendar;
import java.util.Date;

public class DatumPelda{

    private static Calendar naptar = Calendar.getInstance();
    private static Calendar most = Calendar.getInstance();

    public static Date setDatum(int ev, int ho, int nap) {
        naptar.set(Calendar.YEAR, ev);
        naptar.set(Calendar.MONTH, ho);
        naptar.set(Calendar.DAY_OF_MONTH, nap);
        return naptar.getTime();
    }

    public static void main(String[] args){
        System.out.println("Aktuális dátum: " +
                          most.get(Calendar.YEAR) + ". " +
                          (most.get(Calendar.MONTH)+1) + ". " +
                          most.get(Calendar.DAY_OF_MONTH) + "." );

        // Date szulEv = setDatum(2000,Calendar.APRIL +1 ,23);
        Date szulEv = setDatum(2000,4,23);
        naptar.setTime(szulEv);
        System.out.println("Születési dátum: " +
                          naptar.get(Calendar.YEAR) + ". " +
                          naptar.get(Calendar.MONTH) + ". " +
                          naptar.get(Calendar.DAY_OF_MONTH) + ".");
        int kor = most.get(Calendar.YEAR)- naptar.get(Calendar.YEAR);
        System.out.println("Életkora: " + kor + " év");
    } }
```

JAVA OSZTÁLYOK

Jó és rövid (☺) leírás:

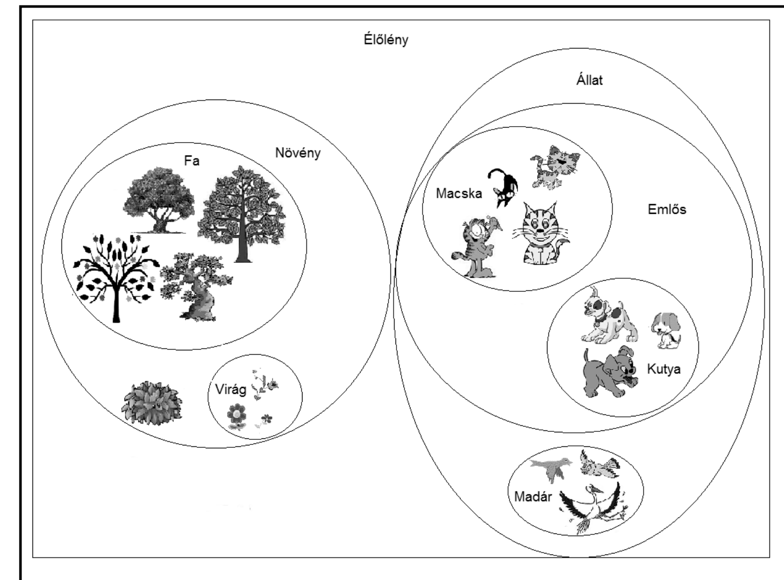
Auth Gábor (volt Pollackos ☺): Java-Forum/Java-suli:

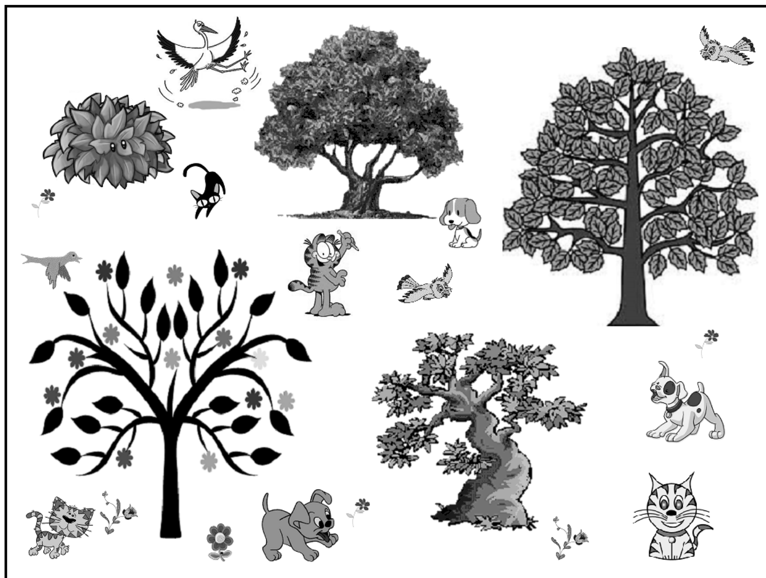
http://wiki.javaforum.hu/display/JAVAFORUM/Java-Suli

```
import java.util.Calendar;
import java.util.Date;

private static Calendar naptar = Calendar.getInstance();
private static Calendar most = Calendar.getInstance();

 public static Date setDatum(int ev, int ho, int nap) {
     naptar.set(Calendar.YEAR, ev);
     naptar.set(Calendar.MONTH, ho);
     naptar.set(Calendar.DAY_OF_MONTH, nap);
     return naptar.getTime();
 }
System.out.println("Aktuális dátum: " +
                  most.get(Calendar.YEAR) + ". " +
                  (most.get(Calendar.MONTH)+1) + ". " +
                  most.get(Calendar.DAY_OF_MONTH) + "." );

// Date szulEv = setDatum(2000,Calendar.APRIL +1 ,23);
Date szulEv = setDatum(2000,4,23);
naptar.setTime(szulEv);
System.out.println("Születési dátum: " +
                  naptar.get(Calendar.YEAR) + ". " +
                  naptar.get(Calendar.MONTH) + ". " +
                  naptar.get(Calendar.DAY_OF_MONTH) + ".");
```

Innentől önálló feldolgozás

(ill. inkább a korábban tanultak önálló átismétlése)





OBJEKTUM-ORIENTÁLT GONDOLKOZÁS, MODELLEZÉS

Az absztrakció révén osztályhierarchiát hozhatunk létre, de dolgozni csak konkrét egyedekkel, vagyis az osztályok példányaival, az objektumokkal tudunk.

Az osztály egy-egy fogalom definiálására szolgál. Leírásakor egy-egy speciális típust határozunk meg, abból a célból, hogy később ilyen típusú változókkal tudjunk dolgozni.

Az Osztaly típusú változó majd Osztaly típusú objektumot tartalmaz.

OBJEKTUM-ORIENTÁLT PARADIGMA

Az osztályokban megadjuk azokat a tulajdonságokat és viselkedéseket, amelyekkel az ebbe az osztályba tartozó objektumokat jellemezni akarjuk. ⇒ A hasonló tulajdonságokkal és viselkedéssel rendelkező objektumok egy osztályba kerülnek.

Az **objektum** az **osztály** egy **példány**a – a **példányosítás** során jön létre.

OBJEKTUM-ORIENTÁLT PARADIGMA

Példányváltozó:
példányonként más-más lehet az értéke.
A példány állapotát írja le.

Egy osztály minden példánya saját adattagokkal rendelkezik. A példányosítás során lefoglalódik az adatoknak megfelelő tárrész. Ahány példány van, annyiszor foglalunk helyet a tárban.

Osztályváltozó (statikus változó):
csak egyetlen példányban létezik

OBJEKTUM-ORIENTÁLT PARADIGMA

Az objektum egy információtároló egység, vannak
tulajdonságai → adatok (mezők)
viselkedése → metódusok
Minden objektum egyedi!

Mezői lehetnek: példányváltozók – osztályváltozók

Metódusai: példánymetódusok – osztálymetódusok

OBJEKTUM-ORIENTÁLT PARADIGMA

Példány metódus:
a példányok viselkedését írja le.

Osztálymetódus (statikus metódus):
objektumok nélkül is tud dolgozni, nem kell példányosítani.
Csak statikus példányváltozókra hivatkozhat.

Egy osztálymetódus az osztályok közötti üzenettel hívható:

Osztaly.osztalyMetodus

OBJEKTUM-ORIENTÁLT PARADIGMA – TULAJDONSÁGOK

Objektum: adattagok + műveletek (metódusok)

Információt tárol, kérésre feladatokat hajt végre.

Belső állapota van, üzeneten keresztül lehet megszólítani.

Felelős feladatainak korrekt elvégzéséért.

Az objektumnak mindig van egy állapota (adattagok pillanatnyi értékei írják le). Két objektumnak ugyanaz az állapota, ha az adattagok értékei megegyeznek.
Az objektum műveleteket hajt végre, melyek hatására állapota megváltozhat, de mindig emlékszik állapotára.

Minden objektum egyértelműen azonosítható.

OBJEKTUM-ORIENTÁLT PARADIGMA – FOGALMAK

Osztály – Példány:

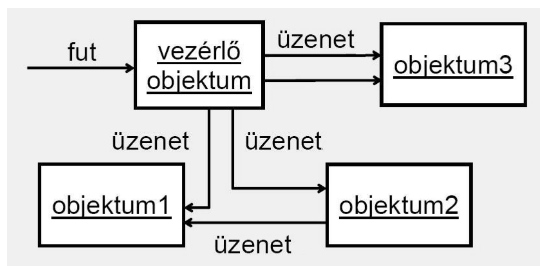
Osztály (class): Olyan objektumminta vagy típus, mely alapján **példány**okat (objektumokat) hozhatunk létre. Minden objektum egy jól meghatározott osztályhoz tartozik.

Objektum életciklusa: „megszületik”, „él”, „meghal”

Az objektumot létre kell hozni, és inicializálni kell!

OBJEKTUM-ORIENTÁLT PARADIGMA – MŰKÖDÉS

Egy objektumorientált program egymással kommunikáló objektumok összessége, melyben minden objektumnak megvan a feladatköre:



OBJEKTUM-ORIENTÁLT PARADIGMA – ALAPELVEK

Az OOP alapelvei:

- egységbezárás;
- öröklődés;
- polimorfizmus (sokalakúság)

- újrahasznosíthatóság

Egységbezárás (**az információ elrejtése**):
Az objektumokat nem lehet kívülről nem várt módon manipulálni. Csak meghatározott metódusokon keresztül módosítható az állapot.

OBJEKTUM-ORIENTÁLT PARADIGMA – ALAPELVEK

Egységbezárás (encapsulation): az adatokat és a hozzájuk tartozó eljárásokat egyetlen egységben kezeljük (osztály)

A feladatok elvégzésének „hogyan”-ja az objektum belügye. Az objektum belseje sérthetetlen. Az objektummal csak az interfészen keresztül lehet kommunikálni.

Az osztály adatai, metódusai csak a konkrét objektumon keresztül érhetők el, az osztály változóit csak a metódusokon keresztül változtathatjuk meg.

JAVA KÓD – OSZTÁLYDEFINÍCIÓ

Fej:

<módosító> class OsztályNév
Pl.: public class Elso

Törzs

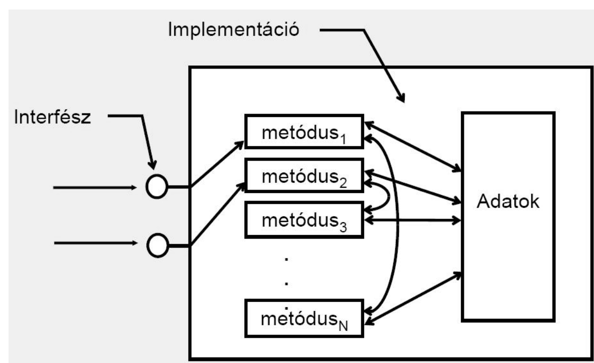
Változók deklarálása
Metódusok deklarálása

Egy osztály módosítói lehetnek:

public: az osztály nyilvánosan hozzáférhető
abstract : az osztály nem példányosítható
final: nem lehet őse más osztálynak

OBJEKTUM-ORIENTÁLT PARADIGMA – ALAPELVEK

Egységbezárás:



JAVA KÓD – OSZTÁLYDEFINÍCIÓ

Egy változó módosítói lehetnek:

Hozzáférési módosítók:

nincs : csomagszintű hozzáférés
public : nyilvánosan hozzáférhető
private : csak az osztályon belül érhető el
protected : az osztályból, az utódokból és az azonos csomagból érhető el

Egyéb módosítók:

static : osztályváltozót jelez
final : konstans, a változó értéke nem változhat

Egy metódus módosítói lehetnek:
ugyanezek + egyéb módosítóként:

abstract : a metódusnak nincs törzse

UML – OSZTÁLYDIAGRAMOK

Jelölések:

+ public
- private
protected

static

FINAL

abstract

JAVA KÓD – OSZTÁLYDEFINÍCIÓ

Kitérő (ismétlés – típusok):

Egyszerű típus: azonosítójával közvetlenül hivatkozunk a változó memóriahelyére. Ezt a helyet a rendszer a deklaráló utasítás végrehajtásakor foglalja le.

Referencia típus: A referencia típusú változók objektumokra mutatnak. Egy referencia típusú változó azonosítójával közvetve hivatkozunk az objektum memóriahelyére. (Maga a hivatkozás rejtve marad.) Deklaráláskor csak a referencia részére foglalunk tárterületet, maga az objektum a példányosítás során jön létre.

JAVA KÓD – OSZTÁLYDEFINÍCIÓ

Objektumok létrehozása: a new operátor segítségével:

OsztalyNev objektum;
objektum = new OsztalyNev(*paraméterlista*);

vagy:

OsztalyNev objektum = new OsztalyNev(*paramlista*);

(A new operátor hatására a rendszer lefoglal egy tárterületet az objektum számára.)

JAVA KÓD – OSZTÁLYDEFINÍCIÓ

Mi „hozza létre" az objektumot? A konstruktor.

Segítségével beállíthatjuk a változók kezdeti értékeit és elvégezhetünk olyan műveleteket, amelyeknek az objektum születésekor automatikusan kell végrehajtódniuk.

Egy osztálynak több konstruktora is lehet.

Definiálása:

```
public Osztalynev(paraméterlista) {
        // inicializálás
}
```

this objektumreferencia:
az aktuális objektumra mutató referencia pl.: this.nev;

JAVA KÓD – OBJEKTUM MEGSZÜNTETÉSE

Mi „szünteti meg” az objektumot? A destruktor helyett: szemétgyűjtés

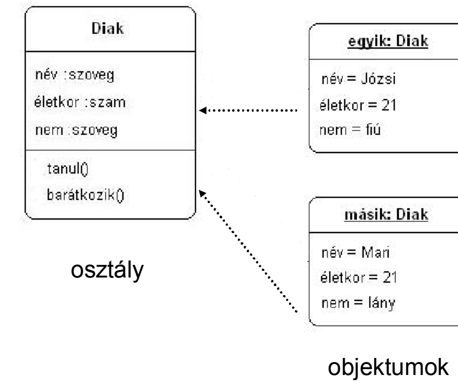
Szemétgyűjtés (garbage collection):

Felszabadítja a memóriát, de pontosan nem lehet tudni, hogy mikor. Dinamikus tárkezelés.

Azokat az objektumokat tünteti el, amelyeket már senki sem használ.

OBJEKTUM-ORIENTÁLT PARADIGMA – JAVA PÉLDA

Egy UML:



osztály

objektumok

JAVA KÓD – OBJEKTUMELÉRÉS

Metódus elérése: pont operátorral

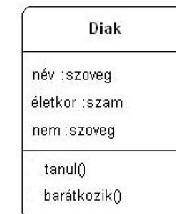
Mező (adattag) elérése: set/get metódussal

Pl.:
Ember egyik=new Ember();

egyik.alszik();
System.out.println(***egyik.getNev()***);
egyik.setHajSzin(”barna”);

OBJEKTUM-ORIENTÁLT PARADIGMA – JAVA PÉLDA

Először az osztályt definiáljuk



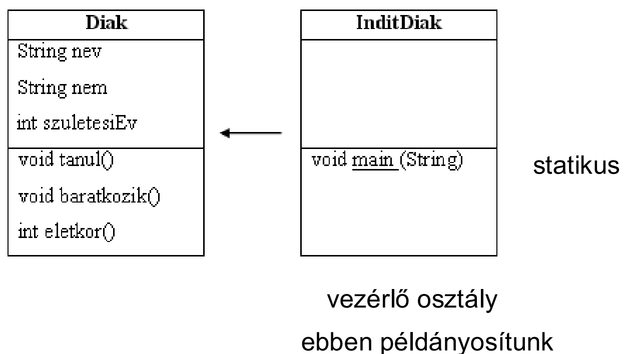
osztálynév: Diak

mezők:
- nev, nem
- szuletesiEv !!!

metódusok:
- tanul()
- baratkozik()
- eletkor() !!!

OBJEKTUM-ORIENTÁLT PARADIGMA – JAVA PÉLDA

Az osztálystruktúra UML diagramja



statikus

vezérlő osztály

ebben példányosítunk

OBJEKTUM-ORIENTÁLT PARADIGMA – MÁSIK PÉLDA

A statikus tagokhoz statikus metódusok tartoznak!

this objektumreferencia helyett OsztalyNev.valtozo módon hivatkozunk! (Osztályon belül is.)

```
public static float getHatar() {
    return atlagHatar;
}

public static void setAtlagHatar(float atlagHatar) {
    Diak.atlagHatar = atlagHatar;
}

public static int getSzorzo() {
    return szorzo;
}

public static void setSzorzo(int szorzo) {
    Diak.szorzo = szorzo;
}
```

```
public class Diak{

   private String nev, nem, hajSzin;
   private int szuletesiEv;
   private static int aktualisEv;

   public Diak(String nev, String nem, int szuletesiEv){
       this.nev = nev;
       this.nem = nem;
       this. szuletesiEv = szuletesiEv;
   }

   public void setHajSzin(String hajSzin){
       this.hajSzin = hajSzin;
   }

   public String getHajSzin(){
       return this.hajSzin;
   }

   public int eletkor(){
       return aktualisEv - szuletesiEv;
   }
...
```

OBJEKTUM-ORIENTÁLT PARADIGMA – PÉLDA

```
public class Diak {

    private String nev, ehaKod;
    private double atlag;
    private static float atlagHatar;
    private static int szorzo;
```

Melyik jó konstruktor?:

```
public Diak(String nev, String ehaKod) {
    this.nev = nev;
    this.ehaKod = ehaKod;
}
```

☺

```
public Diak(String ehaKod, String nev) {
    this.nev = nev;
    this.ehaKod = ehaKod;
}
```

☺ de együtt nem szerepelhetnek

KONSTRUKTOROK

```
public Diak(String nev) {
    this.nev = nev;
}
```

☺ Formája jó, csak nem illeszkedik a feladathoz, mert nincs mód az eha-kód megadására.

```
public Diak() {
}
```

☺ Formája jó, csak nem illeszkedik a feladathoz.

Ettől függetlenül sokszor megírják az üres konstruktort – egy esetleges előre nem tervezett felhasználás kedvéért. Ilyenkor azonban illik gondoskodni a meg nem adott paraméterk default értékéről.

KONSTRUKTOROK

És ha megengedett mindkét példányosítás? (Azaz opcionális, hogy az elején beállítunk-e átlagot vagy sem.)

Egymásba csúszó konstruktorok tervezési minta:

```
public Diak(String nev, String eha) {
    this.nev = nev;
    this.eha = eha;
}
public Diak(String nev, String eha, float atlag) {
    this(nev, eha);
    this.atlag = atlag;
}
```

KONSTRUKTOROK

```
public Diak(String nev, String ehaKod, float atlag) {
    super();
    this.nev = nev;
    this.ehaKod = ehaKod;
    this.atlag = atlag;
}
```

? A feladat szövegétől függ.

```
public Diak(String nev, String ehaKod, int szorzo) {
    this.nev = nev;
    this.ehaKod = ehaKod;
    this.szorzo = szorzo;
}
```

Statikus változó nem lehet konstruktor paramétere!

(És nem this-zel, hanem osztálynévvel hivatkozunk rá.)

KONSTRUKTOROK

Néha fordítva is szokás:

```
public static final float DEFAULT_ATLAG = 0;

public Diak(String nev, String ehaKod, float atlag) {
    this.nev = nev;
    this.ehaKod = ehaKod;
    this.atlag = atlag;
}

public Diak(String nev, String ehaKod) {
    this(nev, ehaKod, DEFAULT_ATLAG);
}
```

Adattag **csak akkor** lehet publikus, ha **static final**!!!

OBJEKTUM-ORIENTÁLT PARADIGMA – PÉLDA

Melyik a jó metódus?

```
public int eladasiAr(){
    return (int) (this.mennyiseg * this.egysegAr *(1 + Ital.afa / 100.));
}
```

vagy:

```
public int eladasiAr(double mennyiseg, int egsegAr){
    return (int) (this.mennyiseg * this.egysegAr *(1 + Ital.afa / 100.));
}
```

```
public int eladasiAr(double mennyiseg, int egysegar){
    return (int)(mennyiseg * egysegar * (1 + Ital.afa / 100.));
}
```

Mikor kell pont a 100 után?

KITÉRŐ – MÁSIK PÉLDA

Melyik a jó metódus:

```
public double terulet(){
    return hosszusag*szelesseg;
}
```

vagy:

```
public double terulet(double hosszusag, double szelesseg){
    return hosszusag*szelesseg;
}
```

Elvileg lehet-e jó a második változat?

Hogyan kellene kijavítani, hogy tényleg jó is legyen?

public **static** double terulet(…)

OBJEKTUM-ORIENTÁLT PARADIGMA – PÉLDA

1. Miért nem jó?

```
public void termekAr(){
    eladasiAr= afa*literenkentiEgysegAr;
}
```

2. Miért nem jó, ha mindent a beolvasás metódusban oldunk meg? (összegzést, max-keresést, stb.)

3. Miért NINCS kiíratás az alaposztályokban?

JAVA OSZTÁLYOK

Jó és rövid (☺) leírás:

Auth Gábor (volt Pollackos ☺): Java-Forum/Java-suli:

http://wiki.javaforum.hu/display/JAVAFORUM/Java-Suli

KITÉRŐ: DÁTUMKEZELÉS

```
import java.util.Calendar;
import java.util.Date;

public class DatumPelda{

    private static Calendar naptar = Calendar.getInstance();
    private static Calendar most = Calendar.getInstance();

    public static Date setDatum(int ev, int ho, int nap) {
        naptar.set(Calendar.YEAR, ev);
        naptar.set(Calendar.MONTH, ho);
        naptar.set(Calendar.DAY_OF_MONTH, nap);
        return naptar.getTime();
    }

    public static void main(String[] args){
        System.out.println("Aktuális dátum: " +
                          most.get(Calendar.YEAR) + ". " +
                          (most.get(Calendar.MONTH)+1) + ". " +
                          most.get(Calendar.DAY_OF_MONTH) + "." );

        // Date szulEv = setDatum(2000,Calendar.APRIL +1 ,23);
        Date szulEv = setDatum(2000,4,23);
        naptar.setTime(szulEv);
        System.out.println("Születési dátum: " +
                          naptar.get(Calendar.YEAR) + ". " +
                          naptar.get(Calendar.MONTH) + ". " +
                          naptar.get(Calendar.DAY_OF_MONTH) + ".");
        int kor = most.get(Calendar.YEAR)- naptar.get(Calendar.YEAR);
        System.out.println("Életkora: " + kor + " év");
    } }
```

```
            import java.util.Calendar;
            import java.util.Date;

   private static Calendar naptar = Calendar.getInstance();
   private static Calendar most = Calendar.getInstance();

    public static Date setDatum(int ev, int ho, int nap) {
        naptar.set(Calendar.YEAR, ev);
        naptar.set(Calendar.MONTH, ho);
        naptar.set(Calendar.DAY_OF_MONTH, nap);
        return naptar.getTime();
    }
System.out.println("Aktuális dátum: " +
                    most.get(Calendar.YEAR) + ". " +
                    (most.get(Calendar.MONTH)+1) + ". " +
                    most.get(Calendar.DAY_OF_MONTH) + "." );

// Date szulEv = setDatum(2000,Calendar.APRIL +1 ,23);
Date szulEv = setDatum(2000,4,23);
naptar.setTime(szulEv);
System.out.println("Születési dátum: " +
                    naptar.get(Calendar.YEAR) + ". " +
                    naptar.get(Calendar.MONTH) + ". " +
                    naptar.get(Calendar.DAY_OF_MONTH) + ".");
```