Programozás III

KOLLEKCIÓK

PROBLÉMAFELVETÉS 1.

A gyakorlaton vett példával kapcsolatban:

Minden résztvevő esetén tároljuk azoknak a rendezvényeknek a listáját, amelyen az illető részt vett.

– Mi a baj?

Redundancia ☹

PROBLÉMAFELVETÉS 1.

Több – több kapcsolat megoldása:

adatbáziskezelésben: kapcsolótábla

Itt pedig próbálkozhatunk kapcsoló-osztállyal:

```
public class KapcsoloOsztaly {
    private int rendezvenySorszam;
    private int resztvevoSorszam;

    public KapcsoloOsztaly(int rendezvenySorszam, int resztvevoSorszam) {
        this.rendezvenySorszam = rendezvenySorszam;
        this.resztvevoSorszam = resztvevoSorszam;
    }
```

PROBLÉMAFELVETÉS 1.

A Rendezvény osztályban:

```
private List<KapcsoloOsztaly> kapcsolok = new ArrayList<>();

public void resztVesz(Resztvevo resztvevo) {
    if (resztvevo.belephet(this)) {
        resztvevo.fizet(resztvevo.getReszveteliDij());
        kapcsolok.add(new KapcsoloOsztaly(this.getEgyediSorszam(),
                                          resztvevo.getEgyediSorszam()));
        resztvevoSzam++;
        bevetel += resztvevo.getReszveteliDij();
    }
}
```

PROBLÉMAFELVETÉS 1.

Egyedi sorszám:

```
private int egyediSorszam;
private static int sorszam;

public Resztvevo(String nev) {
    this.nev = nev;
    sorszam++;
    egyediSorszam = sorszam;
}
```

KITÉRŐ – TÖMBÖK RENDEZÉSE

```
int a[] = new int[length];

//buborek algoritmus
static void buborek(){
   int i, j, temp;
    for (i = 0; i < a.length-1; i++)
        for(j = i+1; j<a.length; j++)
            if (a[j] < a[i]){
                temp = a[i];
                a[i] = a[j];
                a[j] = temp; }
}
```

20000 méretű véletlen tömb esetén:

Buborék: 1375 ms
Arrays.sort: 16 ms

// beépített metódus segítségével:

```
java.util.Arrays.sort(a);
```

PROBLÉMAFELVETÉS 2.

A diákokra vonatkozó kis példát bővítsük úgy, hogy a diákok adatait

a/ névsor szerint növekvő/csökkenő sorrendben

b/ átlag szerint növekvő/csökkenő sorrendben

írassuk ki.

A feladat tömbökkel vagy listával oldható meg.

Arrays (Java Platform SE 6)

static void	**sort**(byte[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of bytes into ascending numerical order.
static void	**sort**(char[] a) Sorts the specified array of chars into ascending numerical order.
static void	**sort**(char[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of chars into ascending numerical order.
static void	**sort**(double[] a) Sorts the specified array of doubles into ascending numerical order.
static void	**sort**(double[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of doubles into ascending numerical order.
static void	**sort**(float[] a) Sorts the specified array of floats into ascending numerical order.
static void	**sort**(float[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of floats into ascending numerical order.
static void	**sort**(int[] a) Sorts the specified array of ints into ascending numerical order.
static void	**sort**(int[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of ints into ascending numerical order.
static void	**sort**(long[] a) Sorts the specified array of longs into ascending numerical order.
static void	**sort**(long[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of longs into ascending numerical order.
static void	**sort**(Object[] a) Sorts the specified array of objects into ascending order, according to the natural ordering of its elements.
static void	**sort**(Object[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements.
static void	**sort**(short[] a) Sorts the specified array of shorts into ascending numerical order.
static void	**sort**(short[] a, int fromIndex, int toIndex) Sorts the specified range of the specified array of shorts into ascending numerical order.
static <T> void	**sort**(T[] a, Comparator<? super T> c) Sorts the specified array of objects according to the order induced by the specified comparator.
static <T> void	**sort**(T[] a, int fromIndex, int toIndex, Comparator<? super T> c) Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.

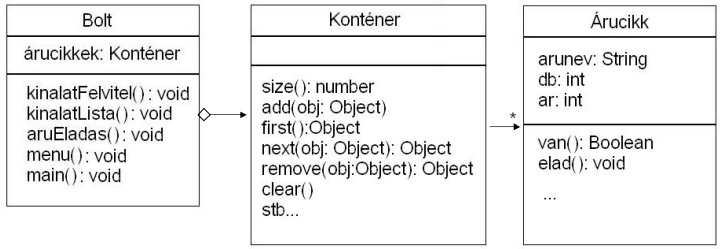
KONTÉNEREK

A konténer olyan objektum, amely objektumokat tárol, és alkalmas különböző karbantartási, keresési és bejárási funkciók megvalósítására.

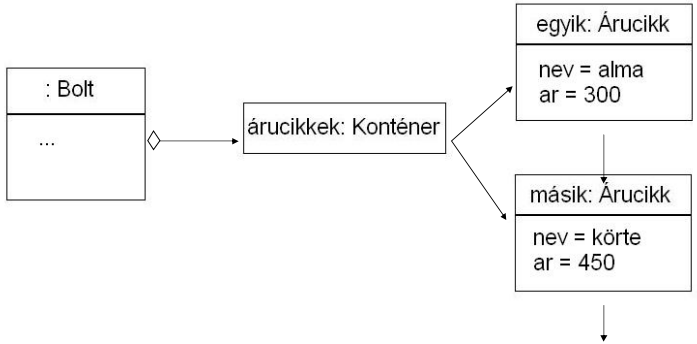
(A tömb is speciális konténer, de nem osztály, nincs viselkedése, vagyis a tömbben tárolt objektumok karbantartására és az elemek keresésére külön meg kell írni az egyes eljárásokat. Egy konténer osztály az elemek tárolásán kívül a különböző keresési, bejárási, karbantartási funkciókat is megvalósítja.)

Egy – sok kapcsolat megvalósítása: konténerek segítségével

OSZTÁLYDIAGRAM



EGYÜTTMŰKÖDÉSI DIAGRAM



Mindegyik elem tudja, hogy ki az utána következő (next()).

KONTÉNEREK – KOLLEKCIÓK

A Java-ban több konténer osztályt implementáltak.
java.util csomag, Collection interfész

A konténerek általánosak, azokba bármilyen objektumot betehetünk.

java.util

Interface Collection<E>

Type Parameters:

E - the type of elements in this collection

All Superinterfaces:

Iterable<E>

All Known Subinterfaces:

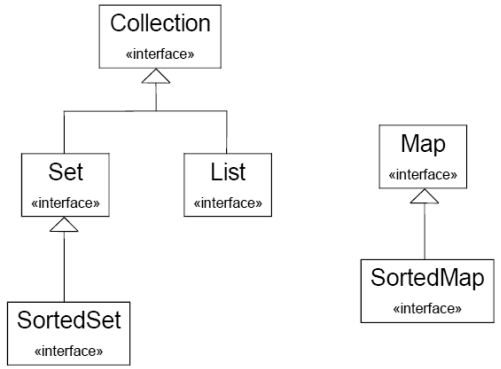
BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

Gyűjtemények ☺

GYŰJTEMÉNY KERETRENDSZER



GYŰJTEMÉNY KERETRENDSZER

Java Collections Framework (Gyűjtemény keretrendszer)

A JCF tartalma:
– **interfészek**: absztrakt reprezentáció, a szolgáltatások megvalósítás-független ábrázolása.

– **implementációk**: az interfészek konkrét implementációi.

– **algoritusok**: a műveleteket megvalósító metódusok. (Ezek többalakú (polimorf) metódusok, vagyis ugyanaz a metódus alkalmazható különböző implementációk esetén.)

GYŰJTEMÉNY KERETRENDSZER

A gyűjtemény interfészei:

Set: nem tartalmaz duplikátumot

List: tartalmazhat duplikátumot
több metódus; bejáráshoz használható az Iterator, ListIterator.

Map: kulcs-érték párokat tartalmazó gyűjtemény
egy kulcshoz csak egy érték tartozhat

GYŰJTEMÉNY KERETRENDSZER

java.util

Interface List<E>

A List interfész és implementációi

Type Parameters:

E - the type of elements in this list

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Implementing Classes:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

LISTA

A lista (List) egy olyan gyűjtemény, amelybe elemeket lehet beszúrni (a végére is és egy adott indexű helyre is), ezeket az elemeket az index alapján el lehet érni, keresni lehet benne, ill. törölni.

A listát implementáló osztályok ezeket a funkciókat valósítják meg, illetve bővítik.

GYŰJTEMÉNY KERETRENDSZER

A List interfész ismert implementációi:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

Érdemes belenézni a megvalósításukba:

NetBeans-ben az osztály nevén Ctrl + kattintás.

http://www.ibm.com/developerworks/java/library/j-jtp07233.html

LISTA

Néhány metódus:

boolean add(Típus elem)
boolean add(int index, Típus elem)
void clear()
boolean contains(Típus elem)
Típus get(int index)
int indexOf(Típus elem)
int lastindexOf(Típus elem)
boolean isEmpty()
boolean remove(Típus elem)
int size()

A LISTÁN VALÓ VÉGIGHALADÁS

1. Közönséges (klasszikus) for ciklus:

Pl.:List<Tipus> adatok = new ArrayList<>();

for(int i = 0; i < adatok.size(); i++) {…}

2. Iteráló for ciklus:

Pl.: List<Tipus> adatok = new ArrayList<>();

for(Tipus adat: adatok) {…}

(Tömbökre is használható.)

Megjegyzés: A szakirodalomban ezt a fajta ciklust részesítik előnyben.

Iterátor példa

```
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorPelda{

   public static void main(String args[]){
       List<String> valami = new ArrayList<String>();
       valami.add("1"); valami.add("2"); valami.add("3");
       valami.add("4"); valami.add("5");

       System.out.println("for-ral");
       for(String s : valami) {
           System.out.println(s);
       }

       System.out.println("iterator-ral");
       Iterator<String> it = valami.iterator();
       while(it.hasNext()) {
           String s = it.next();
           System.out.println(s);
       }
   }
}
```

A LISTÁN VALÓ VÉGIGHALADÁS

3. Iterátor – ennek segítségével is végigmehetünk egy listán, sőt elemet is törölhetünk belőle.

java.util

Interface Iterator<E>

Type Parameters:

E - the type of elements returned by this iterator

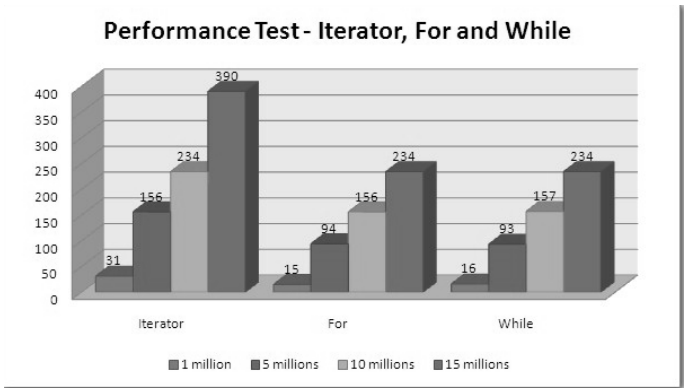
All Known Subinterfaces:

ListIterator<E>, XMLEventReader

All Known Implementing Classes:

BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

IDŐ TESZT



http://www.mkyong.com/java/while-loop-for-loop-and-iterator-performance-test-java/

FELADATMEGOLDÁS

Képezzünk név-szám párosokból álló gyűjteményt, és írassuk ki az elemeit!

A név-szám párosokat úgy tudjuk együtt kezelni és egy listában tárolni, ha objektumokat készítünk belőlük.

Ehhez szükség van a Paros osztály definiálására.

FELADATMEGOLDÁS

```
import java.util.ArrayList;
import java.util.List;
```

```
private List<Paros> adatok = new ArrayList<>();
```

```
private void beolvasas() {
    String nev;
    int szam;

    System.out.print("Az elemek száma: ");
    int n = scanner.nextInt();
    for (int i = 0; i < n; i++) {
        System.out.print("Név: ");
        nev = scanner.next();
        System.out.print("szám: ");
        szam = scanner.nextInt();
        adatok.add(new Paros(nev, szam));
    }
}
```

FELADATMEGOLDÁS

```
public class Paros {

    private String nev;
    private int szam;

    Paros(String nev, int szam) {
        this.nev = nev;
        this.szam = szam;
    }

    String getNev() {
        return this.nev;
    }

    int getSzam() {
        return this.szam;
    }

    @Override
    public String toString() {
        return this.nev + "\t" + this.szam + "\t" + "db";
    }
}
```

FELADATMEGOLDÁS

```
private void kiiratas() {
    System.out.println("\nAz adatok: ");
    for (int i = 0; i < adatok.size(); i++) {
        System.out.println(adatok.get(i));
    }

    //Ugyanez elegásabban:

    for (Paros paros : adatok) {
        System.out.println(paros);
    }
}
```

A FELADAT FOLYTATÁSA

1. Állapítsuk meg, hogy a gyűjtemény tartalmaz-e egy adott elemet!

2. Oldjuk meg a rendezést.

1. FELADAT MEGOLDÁSA

Megoldás:

Nem ugyanazt kellene keresni, csak ugyanolyat.

Erre szolgál az equals és a hashCode metódus.

Ezeket mindig a listában lévő elemekre kell alkalmazni, vagyis abban az osztályban definiálni, amilyen típusú elemeket kezelünk.

1. FELADAT MEGOLDÁSA

```
adatok.add(new Paros("Kati", 5));
System.out.println("\ntartalmazás: " +
                   adatok.contains(new Paros("Kati",5)));
Paros uj = new Paros("Kati",5);
System.out.println("\ntartalmazás: " +
                   adatok.contains(uj));
```

```
tartalmazás: false

tartalmazás: false
```

Magyarázat: A különböző objektumreferenciák.

1. FELADAT MEGOLDÁSA

A Paros osztályban:

Don't panic!

A NetBeans generálja. ☺

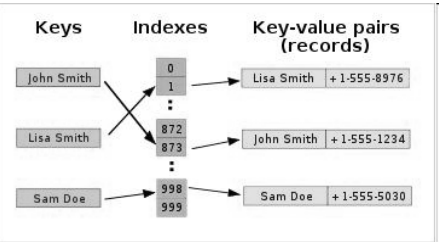
```
@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Paros other = (Paros) obj;
    if (!Objects.equals(this.nev, other.nev)) {
        return false;
    }
    return true;
}

@Override
public int hashCode() {
    int hash = 3;
    hash = 89 * hash + Objects.hashCode(this.nev);
    return hash;
}
```

Most két elemet akkor tekintünk azonosnak, ha megegyezik a nevük.

KITÉRŐ

Hash tábla, hash kód:

A hash tábla kulcs-érték párokat tartalmaz.
Amikor egy elemet elhelyezünk a táblában, akkor megadunk egy kulcsot. Ebből a kulcsból bizonyos hash algoritmus használatával az algoritmus előállít egy memóriacímet (indexet), ahová elhelyezi magának az adatnak a mutatóját.



FELADATMEGOLDÁS (kitérő)

Elvárások a hash kódtól:
1. Egy program futása során akárhányszor hívjuk meg ugyanazt az objektumot, mindig ugyanazt az egész értéket adja vissza. (Ez az érték más-más futtatáskor más-más lehet, de egy futtatás során változatlan.)

2. Ha két objektum az equals(Object) metódus alapján egyforma, akkor a hashCode() metódus eredménye mindkét objektum esetén ugyanaz az egész érték legyen. (Ugyanarra a memóriacímre mutasson.)

FELADATMEGOLDÁS (kitérő)

Hash tábla, hash kód:

A kulcs bármilyen típus lehet, ami nem null, és létezik a hashCode() és equals() metódusa. A gyárilag használt típusok (Object, String, Integer, Boolean, stb.) eleve tartalmazzák ezeket a metódusokat.

http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html#hashCode%28%29

1. FELADAT MEGOLDÁSA

A Paros osztály módosítása után:

```
adatok.add(new Paros("Kati", 5));
System.out.println("\ntartalmazás: " +
                   adatok.contains(new Paros("Kati",5)));
Paros uj = new Paros("Kati",10);
System.out.println("\ntartalmazás: " +
                   adatok.contains(uj));
```

```
tartalmazás: true

tartalmazás: true
```

„SAJÁT" EQUALS() METÓDUS

Természetesen nem muszáj generálni, mi magunk is megírhatjuk az equals() metódust. Pl:

```
public boolean equals(Paros p){
    return (this.nev.equals(p.getNev()) &&
           (this.szam == p.getSzam()));
}
```

(Most akkor tekintjük ugyanolyannak, ha a név is és a szám is egyforma. ☺)

RENDEZÉS A COLLECTIONS OSZTÁLY SEGÍTSÉGÉVEL

Az osztály algoritmusai úgy működnek, hogy páronként összehasonlítják a lista elemeit. Ezért ezek a metódusok megkövetelik, hogy **a konténerbe betett objektumok összehasonlíthatóak legyenek**, vagyis, hogy vagy

a/ maguk implementálják a **Comparable** interfészt, vagy

b/ létezzen hozzájuk a **Compator** interfészt implementáló hasonlító osztály.

RENDEZÉS A COLLECTIONS OSZTÁLY SEGíTSÉGÉVEL

A Comparable interfészt megvalósító osztályok által definiált objektumokból álló kollekciók (gyűjtemények) elemei sorba rendezhetőek, megfordítható a sorrendjük, stb.

Erre szolgálnak a **Collections** osztály metódusai.

A Collections osztály nem példányosítható, statikus metódusai vannak.
(rendezés, megfordítás, minimum-, maximumkeresés, stb.)

RENDEZÉS A COMPARBLE INTERFÉSZ IMPLEMENTÁLÁSÁVAL

Comparable interfész
Ha objektumokat akarunk összehasonlítani, akkor célszerű az objektumokat definiáló osztályt a Comparable interfész megvalósításként (implementációjaként) definiálni.

A java.lang.Comparable interfész egyetlen metódusfejet definiál:

public int compareTo(Object obj)

RENDEZÉS A COMPARBLE INTERFÉSZ IMPLEMENTÁLÁSÁVAL

A **compareTo()** formája kötött, a visszaadott érték
- negatív, ha az objektum kisebb, mint a paraméterként megkapott obj;
- pozitív, ha az objektum nagyobb, mint a paraméterként megkapott obj;
- 0, ha a két objektum egyenlő.

(A String osztály is implementálja a Comparable interfészt ⇒ van benne compareTo() metódus.)

A compareTo() saját magát hasonlítja össze egy kívülről adott objektummal.

RENDEZÉS A COMPARBLE INTERFÉSZ IMPLEMENTÁLÁSÁVAL

A vezérlő osztályban:

```
private void rendezes() {
    System.out.println("Az adatok szám szerint "
                       + "növekvően rendezve: ");
    Collections.sort(adatok);
    for(Paros adat : adatok){
        System.out.println(adat);
    }
}
```

Hogyan lehetne megoldani, hogy egy választástól függően vagy név, vagy szám szerint, vagy növekvő, vagy csökkenő módon rendezzünk?

HF

RENDEZÉS A COMPARBLE INTERFÉSZ IMPLEMENTÁLÁSÁVAL

```
public class Paros implements Comparable<Paros> {

    @Override
    public int compareTo(Paros t) {
        // szám alapján
        if(this.szam == t.szam) return 0;
        if(this.szam > t.szam) return 1;
        return -1;
    }
```

Tömörebben:

```
@Override
public int compareTo(Paros t) {
    // szám alapján
    return (int)Math.signum(this.szam-t.szam);
}
```

MÁS ELVŰ RENDEZÉS

Másik lehetőség:
A Comparator interfészt implementáló hasonlító osztály segítségével rendezünk.

RENDEZÉS A COMPARATOR INTERFÉSZT IMPLEMENTÁLÓ HASONLÍTÓ OSZTÁLY SEGÍTSÉGÉVEL

```
import java.util.Collections;
import java.util.Comparator;

System.out.println("\nNév szerinti rendezés:");
Collections.sort(adatok,new NevSzerint());

for(Paros p : adatok)
        System.out.println(p);

System.out.println("\nSzám szerinti rendezés:");
Collections.sort(adatok,new SzamSzerint());

for(Paros p : adatok)
        System.out.println(p);
```

RENDEZÉS

Rendezések:

Mindegy, hogy melyik fajtát használják, de egy projekten belül lehetőleg csak egyfajta legyen.

(Az AlapOsztaly implements Comparable megoldás esetén is lehet többféle szempontú rendezés – hogyan?)

RENDEZÉS A COMPATOR INTERFÉSZT IMPLEMENTÁLÓ HASONLÍTÓ OSZTÁLY SEGÍTSÉGÉVEL

```
class NevSzerint implements Comparator <Paros> {

   public int compare(Paros p1, Paros p2){
       return p1.getNev().compareTo(p2.getNev());
   }
}

class SzamSzerint implements Comparator <Paros> {

   public int compare(Paros p1, Paros p2){
       return p2.getSzam()-p1.getSzam();
           // mert csökkenő sorrendben szeretnénk
   }
}
```

compare(o1,o2) : Negatív, nulla vagy pozitív értéket ad vissza, attól függően, hogy az első argumentuma kisebb, egyenlő vagy nagyobb, mint a második.

SZUSSZANÁS UTÁN

```
private List<Paros> adatok = new ArrayList<>();
```

Mi ez?

JAVA – GENERIKUSOK

A **generikus** lehetőséget ad osztályok más típussal való paraméterezésére.
Pl.:

java.util

Interface List<E>

Type Parameters:

E - the type of elements in this list

A generikusan (általánosan) definiált List aktuális típus-paramétere az lehet, hogy a lista milyen konkrét típusú adatokat tartalmazzon – pl. List<Paros>

PÉLDA SAJÁT GENERIKUS OSZTÁLYRA

```
public class GenerikusOsszeg <T extends Number> {

   T egyik;
   T masik;

   GenerikusOsszeg(T egyik, T masik){
       this.egyik = egyik;
       this.masik = masik;
   }

   public double osszeg(){
       return (egyik.doubleValue() + masik.doubleValue());
   }

}
```

JAVA – GENERIKUS PÉLDA

De sok más helyen is használhatjuk a generikust.

Pl. készíthetünk olyan saját generikus (általános) számológép típust, amely ugyanúgy teszi a dolgát: összead, szoroz, de egyszer egészekkel, máskor törtekkel, attól függően, hogy az Integer vagy a Double típussal paraméterezve konkretizáltuk-e.

PÉLDA SAJÁT GENERIKUS OSZTÁLYRA

```
public class GenerikusProba{

   public static void main(String args[]){
       int a=2;
       int b=3;
       GenerikusOsszeg<Integer> egy =
                       new GenerikusOsszeg<Integer>(a,b);
       System.out.println("Az összeg: " + egy.osszeg());

       float x = (float)2.4;
       float y = (float)3.2;

       GenerikusOsszeg<Float> ketto =
                       new GenerikusOsszeg<Float>(x,y);
       System.out.println("Az összeg: " + ketto.osszeg());
   }
}
```

MÁSIK PÉLDA SAJÁT GENERIKUS OSZTÁLYRA

```
public static void main(String[] args) {
    Integer a = new Integer(1);
    Integer b = new Integer(2);
    Integer c = nagyobb(a,b);
    Double ad = new Double(2.5);
    Double bd = new Double(1.25);
    Double cd = nagyobb(ad,bd);
    System.out.println("c = " + c + " cd = " + cd);
}

private static <E extends Comparable <E>> E nagyobb(E a, E b) {
    if(a != null && b != null){
        return (a.compareTo(b) > 0)? a : b;
    }else{
        return null;
    }
}
```

```
run:
c = 2 cd = 2.5
```

Enumerátor példa 1.

```
public class EnumPelda {

    public enum Nap{
        HETFO, KEDD, SZERDA, CSUTORTOK,
        PENTEK, SZOMBAT, VASARNAP
    }

    private Nap nap;

    public EnumPelda(Nap nap) {
        this.nap = nap;
    }

    public void milyenNap(){
        switch(nap){
            case KEDD: {
                System.out.println(nap + " a legjobb nap, mert Java előadás.\n");
                break;
            }
            case SZERDA:
            case CSUTORTOK :{
                System.out.println(nap + " is jó, mert Java gyakorlat.\n");
                break;
            }default: System.out.println(nap + ". Mit ér a nap Java nélkül?\n ");
        }
    }
```

MÉG EGY FOGALOM

Enumerátor:
Az enum fix konstans értékek létrehozására használható.
Mivel típusos, így biztonságosabb mint egy int konstans.

Pl: public static final int HETFO = 1

– nem tudjuk, hogy az 1-es mit takar, és kezelni kell az érvénytelen értéket

public enum Nap { HETFO, KEDD,…

– típusos, így nem kell foglalkozni az érvénytelen értékekkel

Lehet konstruktora, és használhatunk benne final és nem final mezőket.

MÉG EGY FOGALOM

```
public static void main(String[] args){
    EnumPelda kedd = new EnumPelda(Nap.KEDD);
    kedd.milyenNap();
    EnumPelda csutortok = new EnumPelda(Nap.CSUTORTOK);
    csutortok.milyenNap();
    EnumPelda pentek = new EnumPelda(Nap.PENTEK);
    pentek.milyenNap();
}
```

```
run:
KEDD a legjobb nap, mert Java előadás.

CSUTORTOK is jó, mert Java gyakorlat.

PENTEK. Mit ér a nap Java nélkül?
```

Enumerátor példa 2.

```
public enum Nap {

    HETFO("Nehezen indul"), KEDD("Végre Java"),
    SZERDA("Hurrá, Java"), CSUTORTOK("Fárasztó"),
    PENTEK("Jön a hétvége :)"),
    SZOMBAT("Hét vége :))"), VASARNAP("Jaj, mindjárt hétfő");

    private String tulajdonsag;

    private Nap(String tulajdonsag) {
        this.tulajdonsag = tulajdonsag;
    }

    public String getTulajdonsag() {
        return tulajdonsag;
    }

    public void setTulajdonsag(String tulajdonsag) {
        this.tulajdonsag = tulajdonsag;
    }
}
```

MÉG EGY FOGALOM

```
run:
A hét napjai:
HETFO tulajdonsága: Nehezen indul
KEDD tulajdonsága: Végre Java
SZERDA tulajdonsága: Hurrá, Java
CSUTORTOK tulajdonsága: Fárasztó
PENTEK tulajdonsága: Jön a hétvége :)
SZOMBAT tulajdonsága: Hét vége :))
VASARNAP tulajdonsága: Jaj, mindjárt hétfő
KEDD tulajdonsága: akkor Belgium
```

MÉG EGY FOGALOM

```
class indito{

    public static void main(String[] args){
        Nap[] napok = Nap.values();
        System.out.println("A hét napjai: ");
        for(Nap nap: napok){
            System.out.println(nap + " tulajdonsága: " +
                                nap.getTulajdonsag());
        }

        Nap nap = Nap.KEDD;
        nap.setTulajdonsag("akkor Belgium");
        System.out.println(nap + " tulajdonsága: "
                + nap.getTulajdonsag());
    }
}
```

ÖSSZETETTEBB ENUM PÉLDA

```
public enum Nap {

    HETFO(new Tulajdonsag("Nehezen indul", -1)),
    KEDD(new Tulajdonsag("Végre Java", 5)),
    SZERDA(new Tulajdonsag("Hurrá, Java", 4)),
    CSUTORTOK(new Tulajdonsag("Fárasztó", 2)),
    PENTEK(new Tulajdonsag("Jön a hétvége !", 3)),
    SZOMBAT(new Tulajdonsag("Hét vége :))", 5)),
    VASARNAP(new Tulajdonsag("Jaj, mindjárt hétfő", 1));

    private Tulajdonsag tulajdonsag;

    private Nap(Tulajdonsag tulajdonsag) {
        this.tulajdonsag = tulajdonsag;
    }

    public Tulajdonsag getTulajdonsag() {
        return tulajdonsag;
    }
}
```

```
class Tulajdonsag{
    private String elnevezes;
    private int minosites;

    public Tulajdonsag(String elnevezes, int minosites) {
        this.elnevezes = elnevezes;
        this.minosites = minosites;
    }

    public String getElnevezes() {
        return elnevezes;
    }

    public int getMinosites() {
        return minosites;
    }

    @Override
    public String toString() {
        return elnevezes + ", minositese: " + minosites;
    }
}
```

ÖSSZETETTEBB ENUM PÉLDA

```
class TulajdonsagSzerint implements Comparator<Nap>{

    @Override
    public int compare(Nap o1, Nap o2) {
        return
            o2.getTulajdonsag().getMinosites() -
                    o1.getTulajdonsag().getMinosites();
    }
}
```

ÖSSZETETTEBB ENUM PÉLDA

```
class indito3 {

    public static void main(String[] args) {
        Nap[] napok = Nap.values();
        System.out.println("A hét napjai: ");
        for (Nap nap : napok) {
            System.out.println(nap + " tulajdonsága: "
                    + nap.getTulajdonsag());
        }
        Arrays.sort(napok, new TulajdonsagSzerint());
        System.out.println("\nRendezve ");
        for (Nap nap : napok) {
            System.out.println(nap + " tulajdonsága: "
                    + nap.getTulajdonsag());
        }
    }
}
```

JAVASLATOK AZ ENUM-HOZ

http://javarevisited.blogspot.hu/2011/08/enum-in-java-example-tutorial.html

http://blog.pengyifan.com/how-to-extend-enum-in-java/

MÉG EGY, AMIT NEM ÁRT ISMÉT MEGBESZÉLNI

```
public class Diak  {

private List<Tantargy> tantargyak = new ArrayList<>();
```

Probléma:

Ha az eredeti listát adjuk vissza a getterben (vagy bármilyen más metódusban), akkor az kívülről módosítható lesz. ☹

MEGOLDÁSOK

Első változat:
A lista = getTantargyak() lista módosítható, de a getTantargyak() hívás eredménye mindig az eredeti lista.

Második változat:
A lista = getTantargyak() lista egyáltalán nem módosítható.

MEGOLDÁSOK

Egy lehetséges megoldás:

```
public List<Tantargy> getTantargyak() {
    return new ArrayList<>(tantargyak);
}
```

Másik lehetőség:

```
public List<Tantargy> getTantargyak() {
    return Collections.unmodifiableList(tantargyak);
}
```

Mi a különbség?