



Java Programozás

2. óra



BITMAN

v: 2020.03.07

5

6

java.lang csomag

- A java.lang csomag alapvető nyelv-közeli és futtatókörnyezet közeli osztályokat és interfészeket tartalmaz, beleértve a gyökér osztályokat is.
- A legfontosabbak a következők:
 - osztály hierarchiát kezelő elemek,
 - nyelv közeli definíciós osztályok,
 - alap kivétel osztályok,
 - matematikai metódusokat biztosító osztályok,
 - szál kezelést biztosító osztályok,
 - biztonsági funkciók,
 - információt visszaadó osztályok
- Automatikusan importálódik, elemeit mindig használjuk

A java.lang
csomag
(részlete)

Object

Runnable

Class

Abstract Class

Final Class

Interface

Boolean

Character

Class

ClassLoader

Number

Math

Process

Runtime

SecurityManager

String

StringBuffer

System

Thread

Throwable

Byte

Double

Float

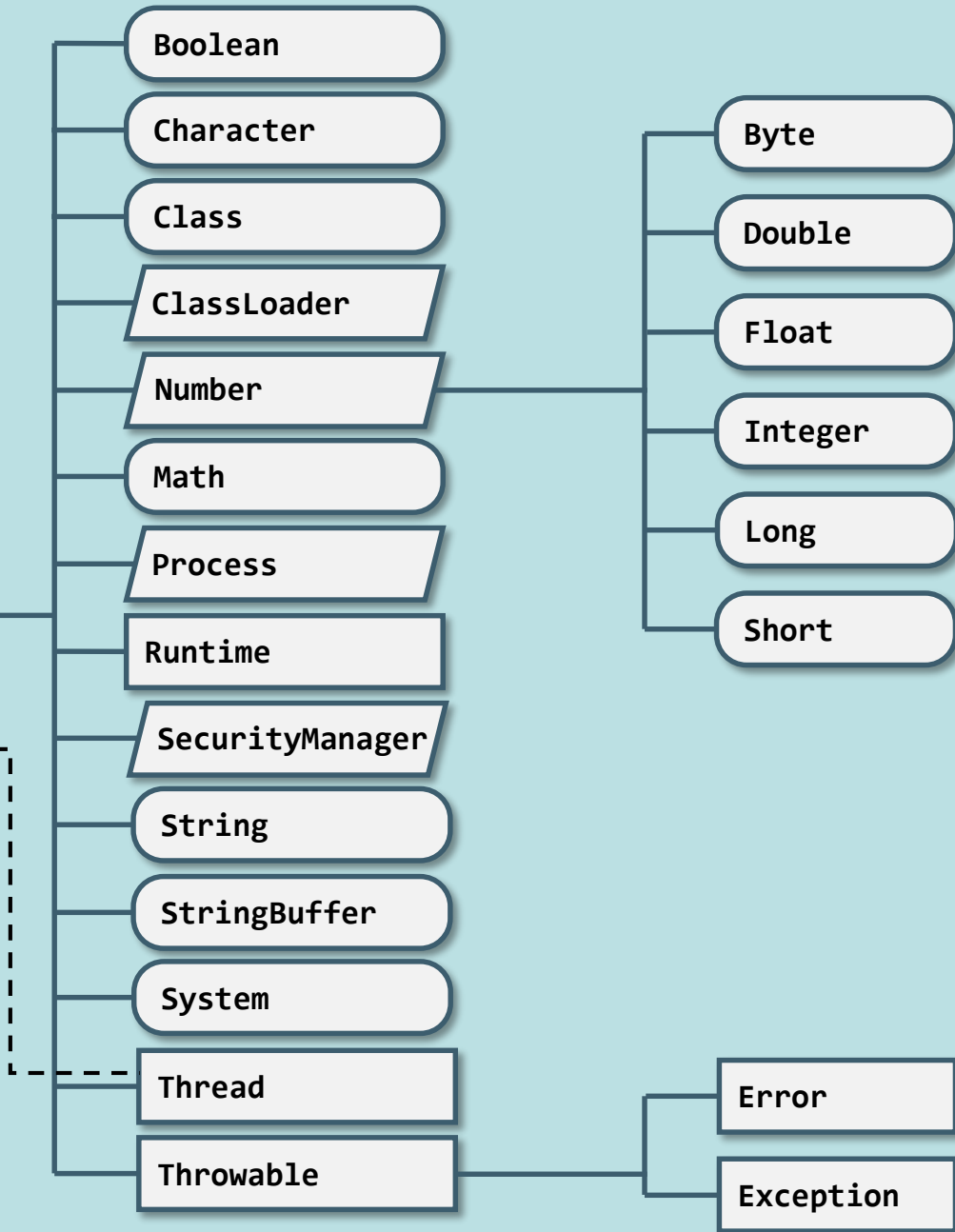
Integer

Long

Short

Error

Exception



java.lang.Object osztály

- Az Object osztály minden osztály közös őse, az osztályhierarchia tetején áll.
- Minden osztály közvetlen vagy közvetett módon utódja az Object osztálynak, így minden osztály rendelkezik az Object osztály metódusaival.
- Ez az osztály definiálja azt az alapvető működést, mely minden objektumnál rendelkezésre áll.
- A java.lang.Object osztálynak része néhány olyan metódus, amelyet szinte mindig használunk:
 - `toString()`
 - `clone()`
 - `equals()`
 - `finalize()`
 - `hashCode()`
 - `getClass()`

java.lang.Object osztály elemei

■ toString() metódus

- Az objektumot szövegesen reprezentálja.
- Minden objektum képes automatikusan sztringgé konvertálódni a segítségével
- Ezt a metódust illik minden új osztály definíciója során felülírni, hogy a megfelelő értéket reprezentálhassa.
- Java alapkódja:

```
public String toString() {  
    return getClass().getName() + "@" +  
        Integer.toHexString(hashCode());  
}
```

```
c:\Java8>java VeremPrg  
Verem@1db9742
```

java.lang.Class osztály

- Minden osztályt futásidőben egy `Class` objektum reprezentál.
- `final` módosítójú osztály
- Használata:
 - Informálódhatunk az osztályról:
 - osztály-e, beágyazott-e, interfész-e, enum-e, ...
 - informálódhatunk a tagjairól
 - dinamikus osztálybetöltés végezhető vele
 - dinamikus példányosítás végezhető vele
 - Hasznos további osztály: `java.lang.reflect`
 - segítségével hozzáférhetünk további metaadatokhoz

java.lang.Class osztály

■ Hasznos metódusok:

- public String getName()
- public Class getSupClass()
- public boolean isInterface()
- public Class[] getInterfaces()
- public Method[] getMethods()
- public Fields[] getFields()
- public Constructor[] getConstructors()

Példakód

```
1. package bitman;
2. public class Emp implements IA{
3.     private String name;
       private int age;

       public Emp(String name, int age){
           this.name = name;
           this.age = age;
4.     }
       public Emp(){
           name = "";
           age = 0;
       }
       public Emp cloneEmp(Emp e){
           Emp x = new Emp("a", 0);
           x.name = e.name;
           x.age = e.age;
           return x;
5.     }
       public String toString(){
           return "Name= "+name+", age= "+age;
       }
       public void Kiir(String x){
6.           System.out.println("Adat= "+x);
       }
}
```



Java **példa ló** osztály:

- csomagban van,
- megvalósított interfész,
- adattagok,
- konstruktorkok,
- metódusok

Írassuk ki az osztály
metaadatait!

Példakód

```
import java.lang.reflect.*;
import java.util.Arrays;
import bitman.Emp;
public class Prg {

    1. static void kiir(String ki){
        System.out.println(ki);
    }
```

```
    public static void main(String args[]) {
```

```
        Emp e = new Emp("Egon", 38);
```

```
        2. Class c = e.getClass();
```

```
        kiir("Type and Name: "+c);
```

```
        3. kiir("Class modifiers: "+Modifier.toString(c.getModifiers()));
```

```
        4. System.out.println("Package name: "+c.getPackage());
```

```
        5. kiir("Is Interface: "+c.isInterface());
```

```
        6. kiir("Is Primitive: "+c.isPrimitive());
```

```
        7. Type[] interfaces = c.getInterfaces();
```

```
        kiir("Number of implemented interfaces: "+interfaces.length);
```

```
        kiir(Arrays.toString(interfaces));
```

```
c:\Java8>java Prg
Type and Name: class bitman.Emp
Class modifiers: public
Package name: package bitman
Is Interface: false
Is Primitive: false

Number of implemented interfaces: 1
[interface bitman.IA]
```

Példakód

```
8. Field[] fields = c.getDeclaredFields();
   kiir("Number of fields: "+fields.length);
   for(Field f : fields){
9.       Class t = f.getType();
10.      kiir("field name: "+f.getName()+" , type: "+t);
   }
11. Constructor<?>[] constructors = c.getConstructors();
   kiir("Number of constructors: "+constructors.length);
   kiir(Arrays.toString(constructors));
12. Method[] methods = c.getDeclaredMethods();
13. //Method[] methods = c.getMethods();
   kiir("Number of declared methods: "+methods.length);
   kiir(Arrays.toString(methods));
}
```

```
Number of fields: 2
field name: name , type: class java.lang.String
field name: age , type: int

Number of constructors: 2
[public bitman.Emp(java.lang.String,int), public bitman.Emp()]

Number of declared methods: 3
[public java.lang.String bitman.Emp.toString(),
public bitman.Emp bitman.Emp.cloneEmp(bitman.Emp),
public void bitman.Emp.Kiir(java.lang.String)]
```

A char primitív típus

- **char** – primitív adattípus, egyetlen karakter tárolására
 - Értékadás: `char a = 'a'; char one = '1'; char space = ' '`;
 - Metódusok:
 - `toCharArray()`, `indexOf()`, `replace()`, `valueOf()`, `charAt()`
 - Az egyes karakterekhez az ASCII kódnak megfelelően számok (sorszám) vannak rendelve, ezt a megfeleltetést használhatjuk a Java programban is:
 - `int a = 'a'; //a=97`
 - `char c = (char)(106); //c='j'`
 - Mivel a char típus nem osztály, további metódusokat nem tudunk rajtuk alkalmazni.

java.lang.Character osztály

- A **Character** osztály a char burkoló (csomagoló) osztálya
- Akkor használjuk, ha objektum szükséges, például:
 - amikor átadunk egy karakter értéket egy metódusnak, ami megváltoztatja az értéket,
 - amikor egy karakter értéket helyezünk el egy adattárolóban, pl: egy ArrayList-ben, ami objektumokat tud csak tárolni, primitív értékeket nem.
- Példányosítás:
 - `Character c1 = new Character('a');`
 - `Character c2 = 'a';`
 - Ilyenkor is létrejön egy objektum, és annak a referenciája lesz a változó értéke!

java.lang.Character osztály

■ Hasznos metódusok:

- `toString()`, `equals()`, `hashCode()`
- `int compareTo()` – összehasonlítja két karaktert
 - 0 – egyformák, < 0 – első kisebb, 0 > - első nagyobb
- `boolean isUpperCase()` – nagybetű?
- `char toUpperCase()` – nagybetűvé alakít
- `boolean isLowerCase()` – kisbetű?
- `char toLowerCase()` – kisbetűvé alakít
- `boolean isDigit()` – számjegy?
- `boolean isLetter()` – betű?
- `boolean isLetterOrDigit()` – betű vagy számjegy?
- `boolean isWhitespace()` – nem látható karakter?
- `boolean isSpaceChar()` – szóköz?

java.lang.String osztály

- Karakteres információk tárolására, kezelésére
 - A String osztály konstans sztringeket tud kezelni, a létrehozott string nem módosítható
 - A String osztály a karakterláncot egy karakter típusú tömbben, a tömb méretét (hosszát) egy számban tárolja
 - final osztály
 - A "szöveg" hatására is létrejön egy String objektum és annak a referenciája található a helyén. Ezért pl. "szöveg".length() helyes.
 - A String objektumokra létezik a + operátor, mint összefűzés.
 - Minden objektum Stringgé konvertálható a toString() metódusának meghívásával.

java.lang.String osztály

■ Létrehozása:

1. `String s1 = new String("Holnapután");`
2. `String s2 = "Tegnap";`
3. `int a=3, b=5;`
`String s3 = "Pont (" +a+ ", " +b+ ")";`
4. `char ch[] = {'B', 'i', 't', 'M', 'a', 'n'};`
`String s4 = new String(ch);`
`String s5 = new String(ch, 3, 3); //s5 = "Man"`
5. `byte data[] = {65, 66, 76, 65, 75};`
`String s6 = new String(data); // s6 = "ABLAK"`

java.lang.String osztály

■ Hasznos metódusok:

- `toString()`, `equals()`, `hashCode()`
- `int length()` – string hossza
- `int compareTo()` – összehasonlítja két sztringet
 - 0 – egyformák, < 0 – első kisebb, 0 > - első nagyobb
- `int compareToIgnoreCase()` – összehasonlítás, a kis- és nagybetűk megkülönböztetése nélkül (`a==A`)
- `equalsIgnoreCase()` – összehasonlítás, a kis- és nagybetűk megkülönböztetése nélkül (`a==A`)
- `char[] toCharArray(s)` – stringből karaktertömb
- `String toLowerCase()` – kisbetűvé alakít
- `String toUpperCase()` – nagybetűssé alakít

java.lang.String osztály

■ Hasznos metódusok:

- `char CharAt(i)` – string *i*. karaktere
- `int indexOf(s)` – *s* előfordulásának kezdő sorszáma
- `int lastIndexOf(s)` – *s* utolsó előfordulásának sorszáma
- `String replace('a','b')` – a kicserélése *b*-re
- `trim()` – szóköz levágása jobbról
- `String substring(k,z)` – rész string *k*-tól *z*-ig
- `boolean startsWith(s)` – *s*-el kezdődik?
- `boolean endsWith(s)` – *s*-re végződik?
- `String concat(s)` – összefűzés (+)
- `boolean isEmpty()` – üres a string?

java.lang.StringBuffer (Builder) osztály

- Két alaposztály van a Javában a kialakulása óta, a String és a StringBuffer
 - A String osztályban olyan stringeket tárolunk, melyek értéke nem fog változni.
 - A StringBuffer osztályt akkor használjuk, ha a szövegen szeretnénk változtatni, ezt elsősorban dinamikus karakterlánc készítésekor (pl. fájlból olvasás)
 - A StringBuffer-ek használata biztonságos több szálas környezetben.
- A StringBuilder osztályt a JDK 5.0-tól vezették be, ami gyorsabb, mint a StringBuffer, de csak egy szálon használható biztonságosan.
- Mindhárom osztály stringek kezelését szolgálja

java.lang.StringBuffer (Builder) osztály

■ Konstruktorok:

- StringBuffer() – üres példány létrehozása
 - Kapacitása mindig 16 karakter!
- StringBuffer(String) – létrehozás szöveges alapértékkel
- StringBuffer(int) – létrehozás a kapacitás megadásával

■ Fontos: a tárolt string és a tároló buffer mérete (hossza) eltérhet. A buffer lehet hosszabb, mint a string!

```
StringBuffer sb1 = new StringBuffer();  
sb1.append("Java is cool");  
int c = sb1.capacity();  
int l = sb1.length();  
  
System.out.println("Capacity: "+c);  
System.out.println("Length: "+l);
```

```
c:\Java8>java SBPrg  
Capacity: 16  
Length: 12
```

java.lang.StringBuffer (Builder) osztály

■ Hasznos metódusok:

- `int capacity()` – a buffer kapacitása (mérete)
- `int length()` – a tárolt string hossza
- `append(d)` – hozzáfűzés (d: tetszőleges típus lehet)
- `insert(i, d)` – beszúrás i.-helytől, d adatot (tetsz. tip.)
- `delete(s, e)` – törlés, s: startindex, e: endindex
- `deleteCharAt(i)` – karakter törlés, i: index
- `replace(s, e, d)` – csere, s: startindex, e: endindex, d: adat
- `reverse()` – tartalom megfordítása →(molatrat)
- `char CharAt(i)` – tartalom i. karaktere
- `setCharAt(i, ch)` – i. karakter lecserélése ch-ra

java.lang.StringBuffer (Builder) osztály

■ Hasznos metódusok:

- `setLength(n)` – string hosszának beállítása, n: new length
- `void trimToSize()` – kapacitás csökkentése a string hosszára
- `void ensureCapacity(m)` – minimális kapacitás megadása
- `void getChars(s,e,d,b)` – tartalom másolása karakter tömbbe
 - s: sourceStartIndex, e: sourceEndIndex
 - d: destinationArray, b: destinationBeginIndex
- `int indexOf(s)` – s előfordulásának indexe
- `int lastIndexOf(s)` – s utolsó előfordulásának indexe

java.lang.Math (StrictMath) osztály

- Matematikai konstansokat, függvényeket tartalmaz
 - minden tagja statikus, hívásuk ezért mindig osztálynévvel
 - pl: `Math.sqrt(x)`;
 - final osztály, ezért nem példányosítható.
- A `java.lang.StrictMath` majdnem ugyanazokat a tagokat tartalmazza, csak az algoritmus, amivel számol szabványos, tehát más nyelvű programokkal azonos eredményt ad.
- A `Math` nem feltétlenül ad szabványos eredményt, de picivel pontosabb.

java.lang.Math (StrictMath) osztály



■ Hasznos konstansok:

- PI – π értéke
- E – Euler-féle szám (e)

■ Hasznos metódusok:

- `abs(x)` – abszolút érték, x lehet: `double`, `float`, `int`, `long`
- `double ceil(double)` – felfelé kerekít
- `double floor(double)` – lefelé kerekít
- `double rint(double)` – a közelebbi egészre kerekít
- `long round(double)` – a közelebbi egészre kerekít
- `int round(float)` – a közelebbi egészre kerekít
- `t min(t, t)` – két szám közül a kisebb
- `t max(t, t)` – két szám közül a nagyobb
 - t lehet: `double`, `float`, `int`, `long`

java.lang.Math (StrictMath) osztály

■ Hasznos metódusok:

- double exp(double) – exponenciális érték
- double log(double) – természetes alapú logaritmus
- double pow(double, double) – hatvány
- double sqrt(double) – négyzetgyök
- double ???(double) – trigonometriai függvények
 - sin, cos, tan, asin, acos, atan, atan2, sinh, cosh
- double random() – 0-1 közötti véletlen szám

```
int rnd = 0;
for (int i=0; i<6; i++){
    rnd = (int)(46*Math.random());
    System.out.println((i+1)+". nyerőszám: "+rnd);
}
```

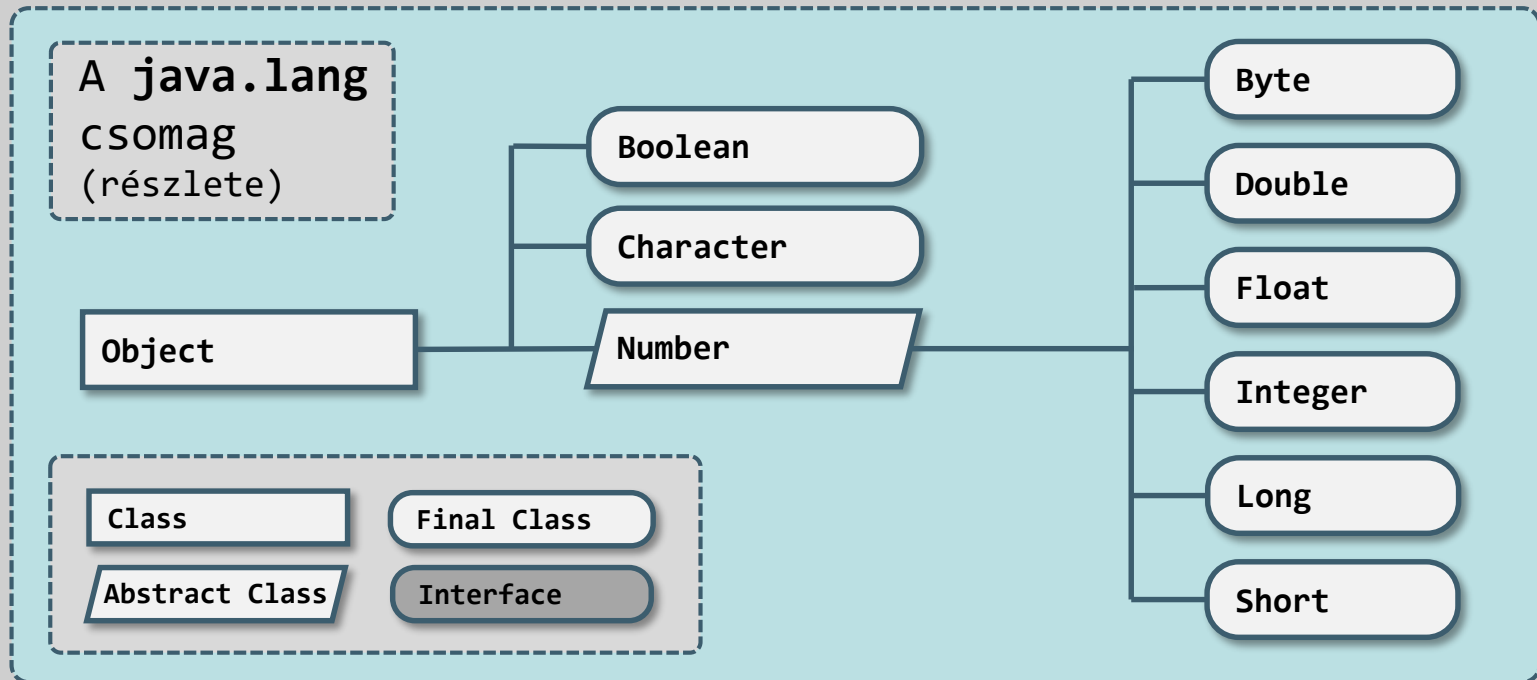
```
c:\Java8>java RndPrg
1. nyerőszám: 28
2. nyerőszám: 20
3. nyerőszám: 36
4. nyerőszám: 13
5. nyerőszám: 33
6. nyerőszám: 1
```

Kitérő: java.util.Random osztály

- `import java.util.Random;`
- A `Random` osztályt példányosítani kell használat előtt:
 - `Random rnd = new Random();`
- Metódusok:
 - `int nextInt(n)` – egész generálása $0 \leq x < n$ tartományban. Nulla lehet, n értéke nem!
 - `int nextInt()` – egész generálása
 - Tartomány: $0 \leq x < \text{Integer.MAX_VALUE}$
 - `long nextLong()` – T: $0 \leq x < \text{Long.MAX_VALUE}$
 - `float nextFloat()` – T: $0 \leq x < 1.0$
 - `double nextDouble()` – T: $0 \leq x < 1.0$
 - `boolean nextBoolean()` – true or false

Csomagoló osztályok

- Minden elemi típusnak van Osztály megfelelője
 - Boolean, Character
 - Byte, Double, Float, Integer, Long, Short
- Szerepük: az elemi típusok használhatók legyenek, olyan helyen, ahol referencia típus szükséges



Példa

```
public class Program{
    public static void main(String args[ ]) {
        int x = 3;
        String s = x.toString();
        System.out.println("s= "+s);
    }
}
```

Program.java:4: error: int cannot be dereferenced

```
String s = x.toString();
```

^

1 error

```
public class Program{
    public static void main(String args[ ]) {
        Integer x = 3;
        String s = x.toString();
        System.out.println("s= "+s);
    }
}
```

s= 3

6

7

A java.util csomag

Tartalma:

- Gyűjtemények
- Property, preferencia
- Dátum, idő kezelés
- Nemzetköziség
- Erőforrás kezelés
- Logolás
- Egyebek...



Gyűjtemény keretrendszer

- A gyűjtemények -tárolók, konténerek, kollekciók- olyan típuskonstrukciós eszközök, melynek célja egy vagy több típusba tartozó objektumok memóriában történő összefoglaló jellegű tárolása, manipulálása és lekérdezése.
- A gyűjtemény keretrendszer (*Java Collections Framework*, JCF) egy egységes architektúra, ami a gyűjtemények használatára és manipulálására szolgál.

Gyűjtemény keretrendszer

■ Tartalma:

- Öt adatmodell (**halmaz**, **lista**, **sor**, **kétvégű sor**, **map**) műveleteit leíró interfészeket,
- Ezen modellek néhány konkrét megvalósítását,
- Bejárókat (iterátorok),
- Néhány algoritmust,
- Az adatszerkezetek szinkronizált használatához szükséges módozatokat.

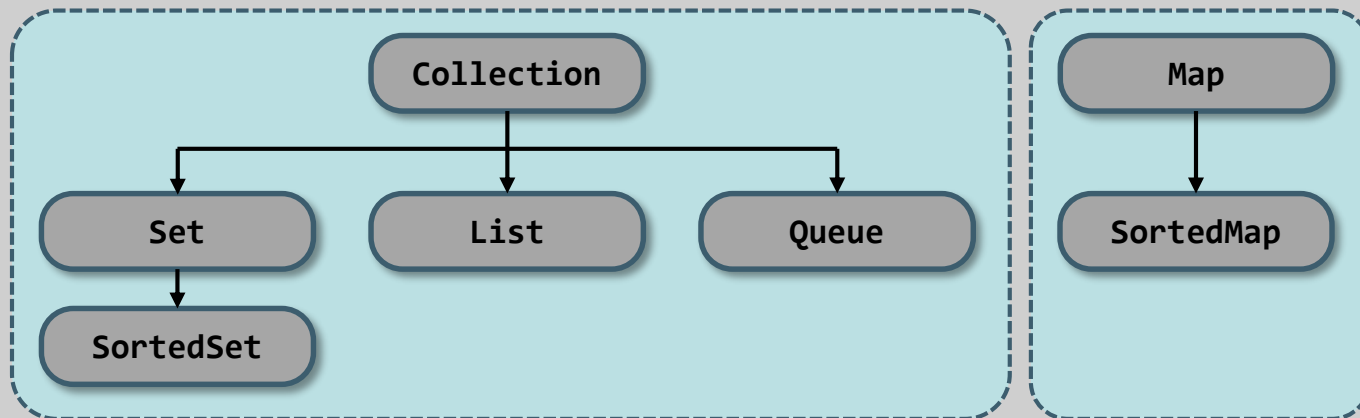
■ A használhatóság érdekében az összes interfész, illetve ezek implementációja generikus típus.

- Ez egyben azt is jelenti, hogy elemi típust csak becsomagolva tudunk gyűjteményben tárolni.

Gyűjtemény keretrendszer

■ Collection interface:

- A map-ek kivételével a többi adatmodellt megvalósító interfészek közös őse.
- Közvetlen megvalósítása nincs, csak abstract implementációi vannak.
- Nem minden metódusát kötelező megvalósítani.
- Azok a metódusai, amelyek opcionálisak úgy vannak deklarálva, hogy dobhatnak `UnsupportedOperationException`-t.



Gyűjtemény keretrendszer

■ Collection interface tartalma:

– Nem opcionális elemek:

- Szeretnék ha az implementáció készítők felüldefiniálnák:
 - `boolean equals(Object o)`
 - `int hashCode()`
- Elemek száma: `int size()`
- Üres-e: `boolean isEmpty()`
- Tartalmazás vizsgálat:
 - `boolean contains(Object element)`
- Iterátor készítés: `Iterator<E> iterator()`
- Elemek tömbként kinyerése:
 - `Object[] toArray()`
 - `<T> T[] toArray(T[] a)`

Gyűjtemény keretrendszer

■ Halmaz (Set) interface

- Nem tartalmazhat duplikált elemeket.
 - Azaz nem lehet két olyan eleme, amelyre igaz, hogy `e1.equals(e2)`.
 - Az `add` metódus hamisat ad ha már van olyan elem a halmazban.
- Tartalma: ugyanaz, mint `Collection` interface-é.
- A halmazműveletek:
 - Elem betétele (`add`),
 - kivétele (`remove`),
 - tartalmazás vizsgálat (`contains`),
 - részhalmaz vizsgálat (`containsAll`),
 - unió (`addAll`),
 - metszet (`retainAll`),
 - különbség (`removeAll`)

Példakód – HashSet

```
import java.util.HashSet;
import java.util.Iterator;
public class Util_1 {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<String>();
        set.add("alma");
        set.add("barack");
        set.add("ananász");
        set.remove("barack");    //equals alapján keres
        set.add("szilva");
        set.add("málna");
        set.add("alma");        //nem rakja bele, false vissza

        System.out.println("List 1:");
        for (String elem : set) System.out.println(elem);
    }
}
```

```
List 1:
szilva
alma
ananász
málna
```

Példakód – HashSet

```
System.out.println("List 2:");
Iterator<String> it = set.iterator();

while (it.hasNext()) System.out.println(it.next());
String[] st = new String[set.size()];    //elemek tömbbe rakása
set.toArray(st);
System.out.println("List 3:");
for (String elem : st) System.out.println(elem);

it = set.iterator();
System.out.println("Delete:");
while (it.hasNext()) {
    String elem = it.next();
    if (elem.startsWith("a")) it.remove();
}
System.out.println("List 4:");
for (String elem : set) System.out.println(elem);
}
```

List 2:
szilva
alma
ananász
málna

List 3:
szilva
alma
ananász
málna

Delete:
List 4:
szilva
málna

Példakód – ArrayList

```
import java.util.ArrayList;

public class ArrayListExample {
    public static void main(String args[]) {
        ArrayList<String> arrL = new ArrayList<String>();
        arrL.add("Albertin");
        arrL.add("Helka");
        arrL.add("Cinnia");
        arrL.add("Stella");
        arrL.add("Abdon");

        System.out.println("Current elements:"+arrL);
        System.out.println("Current 2. element:"+arrL.get(1));

        System.out.println("\nIndex of Cinnia: "+arrL.indexOf("Cinnia"));
        System.out.println("Helka is in array: "+arrL.contains("Helka"));

        arrL.set(2, "Flamina");
        System.out.println("Current elements:"+arrL);
```

```
Current elements:[Albertin, Helka, Cinnia, Stella, Abdon]
Current 2. element:Helka
```

```
Index of Cinnia: 2
Helka is in array: true
Current elements:[Albertin, Helka, Flamina, Stella, Abdon]
```

Példakód – ArrayList

```
Current elements:[Riza, Jarmila, Albertin, Helka, Flamina, Stella, Abdon]  
Current 2. element:Jarmila
```

```
arrL.add(0, "Riza");  
arrL.add(1, "Jarmila");  
System.out.println("\nCurrent elements:"+arrL);  
System.out.println("Current 2. element:"+arrL.get(1));  
  
arrL.remove("Damáz"); //Nem dob hibát!  
arrL.remove("Helka"); //Indexek módosulnak!  
System.out.println("\nCurrent elements:"+arrL);  
System.out.println("Current 2. element:"+arrL.get(1));
```

```
Current elements:[Riza, Jarmila, Albertin, Flamina, Stella, Abdon]  
Current 2. element:Jarmila
```

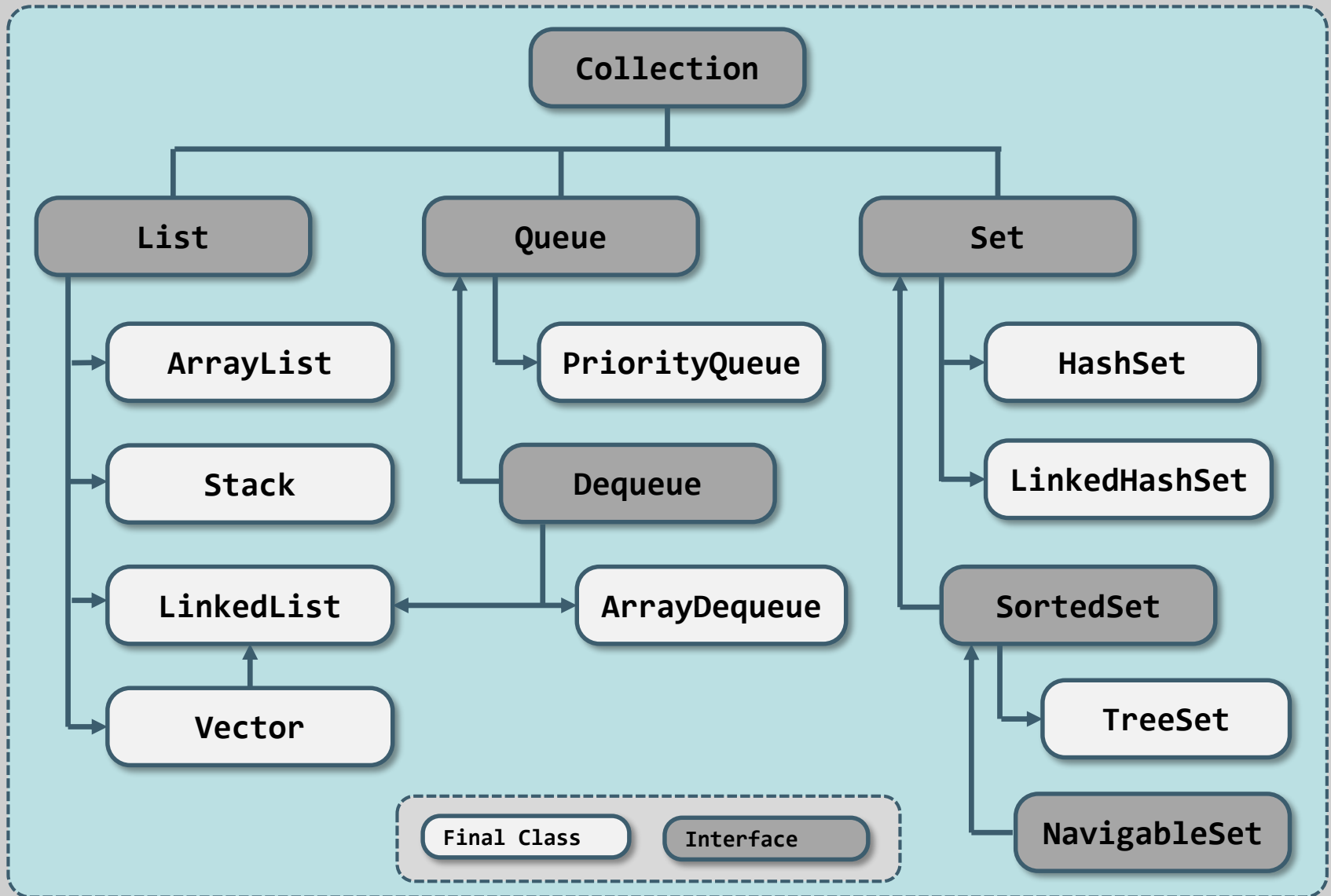
```
arrL.remove(1);  
System.out.println("\nCurrent elements:"+arrL);  
System.out.println("Current 2. element:"+arrL.get(1));
```

```
Current elements:[Riza, Albertin, Flamina, Stella, Abdon]  
Current 2. element:Albertin
```

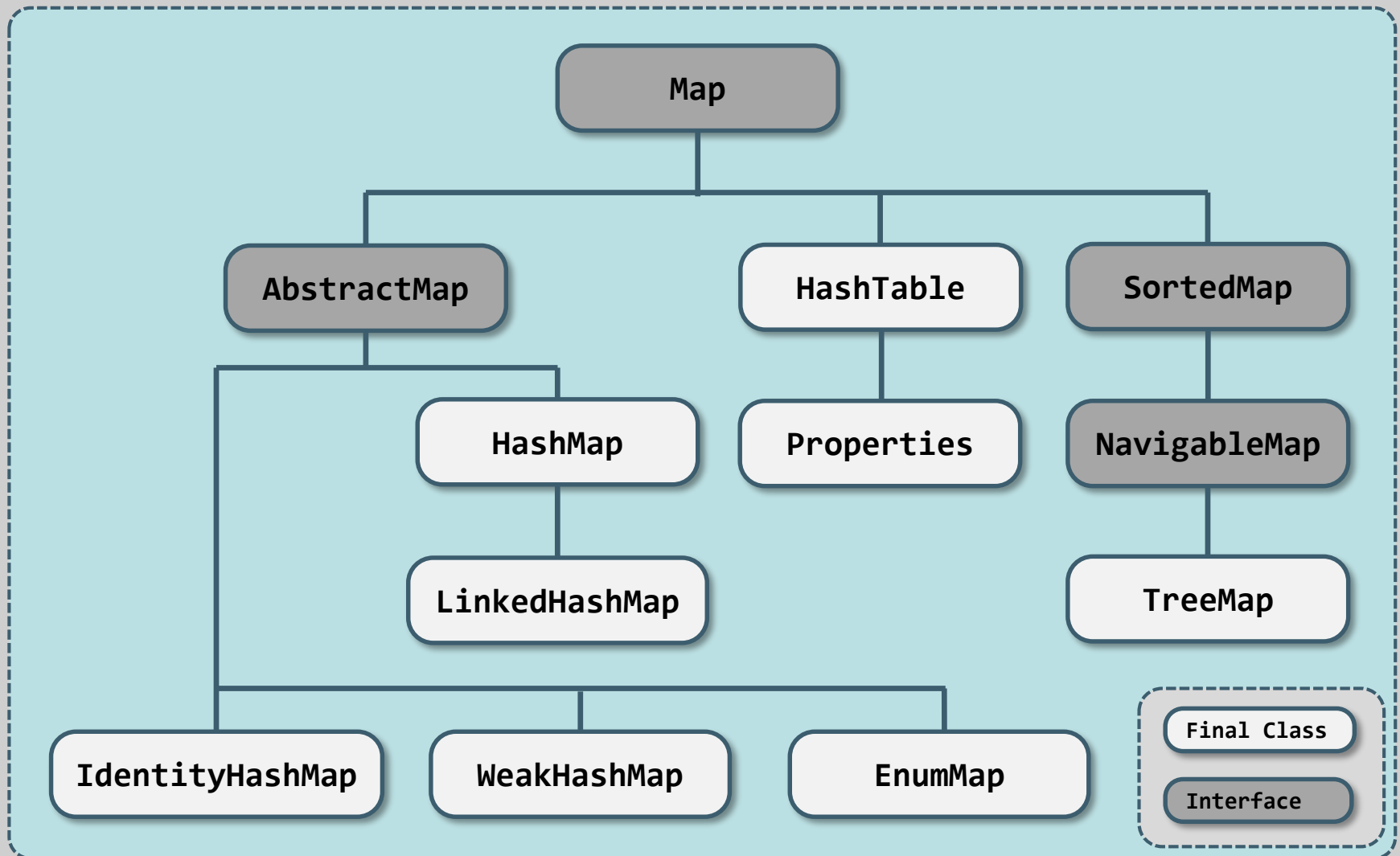
```
for (int j = 0; j < arrL.size(); j++)  
    if (j == 1 || j == 3) arrL.remove(j); //Hibás logika!!  
System.out.println("\nCurrent elements:"+arrL);
```

```
Current elements:[Riza, Flamina, Stella]
```

Gyűjtemény keretrendszer



Java Map keretrendszer



További osztályok a Util csomagban

- Date – Dátum időpont kezelés,
- Locale – Olyan módszerek, melyekkel szoftvereket viszünk át (adaptálunk) más, nem belföldi környezetbe, főként más országokba és kultúrákba.
- Logging – Egy program naplózásával kapcsolatos megoldások

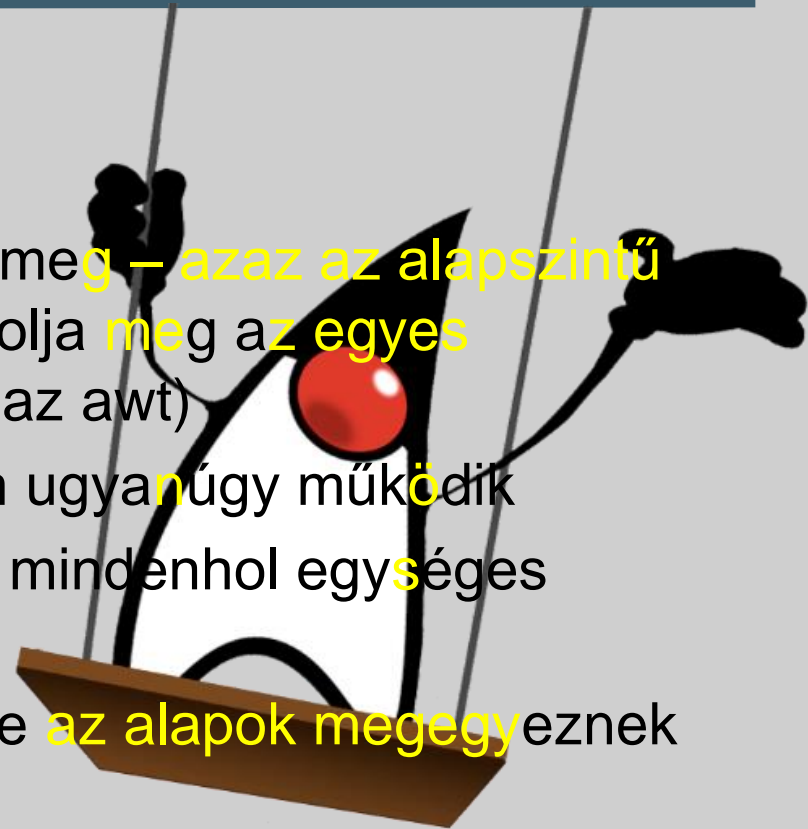
7

8

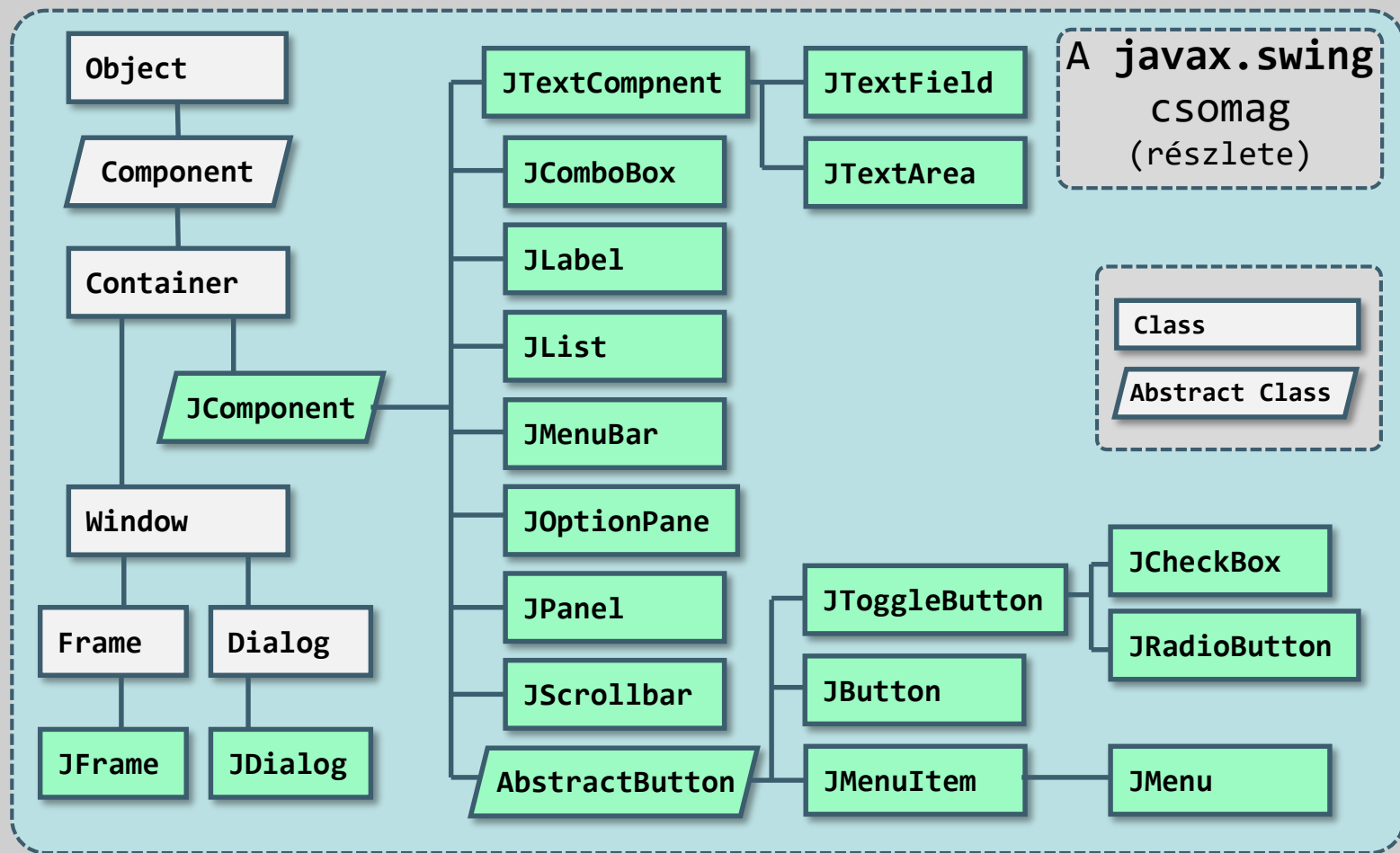
Grafikus felület Javaban

■ Swing

- javax.swing csomag
- Amit lehet, Java kóddal valósít meg – azaz az alapszintű rajzoló utasításokkal maga rajzolja meg az egyes elemeket (emiat lassabb, mint az awt)
- Minden környezetben pontosan ugyanúgy működik
- A host kinézetére hasonlító, de mindenhol egységes megjelenést ad
- Fejlettebb funkciókat biztosít, de az alapok megegyeznek az AWT-vel
- Eseménykezeléshez az AWT biztosította lehetőségeket használja (ill. egészíti ki)
- A J2SE 1.2 verzióban jelent meg



A javax.swing csomag lényegi része



Példa

Melós nyilvántartó

Betöltés

Forrás: Válasszon!

Lista

Adatok száma:

Új adat

Módosítás

Törlés

Kiírás

IQ statisztika

Bezár

Keresés

☒ Kód ☐ Név ☐ Lakóhely ☐ IQ

Kód: =X X=

Keresés

Melós nyilvántartó

Betöltés

Forrás: Válasszon!

Lista

Adatok száma:

Új adat

Módosítás

Törlés

Kiírás

IQ statisztika

Bezár

Keresés

☒ Kód ☐ Név ☐ Lakóhely ☐ IQ

Kód: =X X=

Keresés

Válasszon!
Helyi .dat fájl
Helyi .xml fájl
Helyi .csv fájl
SQLite DB
Web: JSON fájl

8

9

Java fájlkezelés

- A Java nyelvben a fájlkezelés Stream-eken keresztül valósul meg
- Stream: adatok sorozata, egyik végén befelé a másik végén kifelé "folynak" az adatok
- A `java.io` csomag tartalmazza a szükséges osztályokat
- Stream jellemzői:
 - szekvenciális írás, olvasás
 - különböző input források és output nyelők egységesen kezelhetők
 - egy csatorna bemenete lehet egy másik csatorna kimenete is azaz szűrő csatorna

Stream

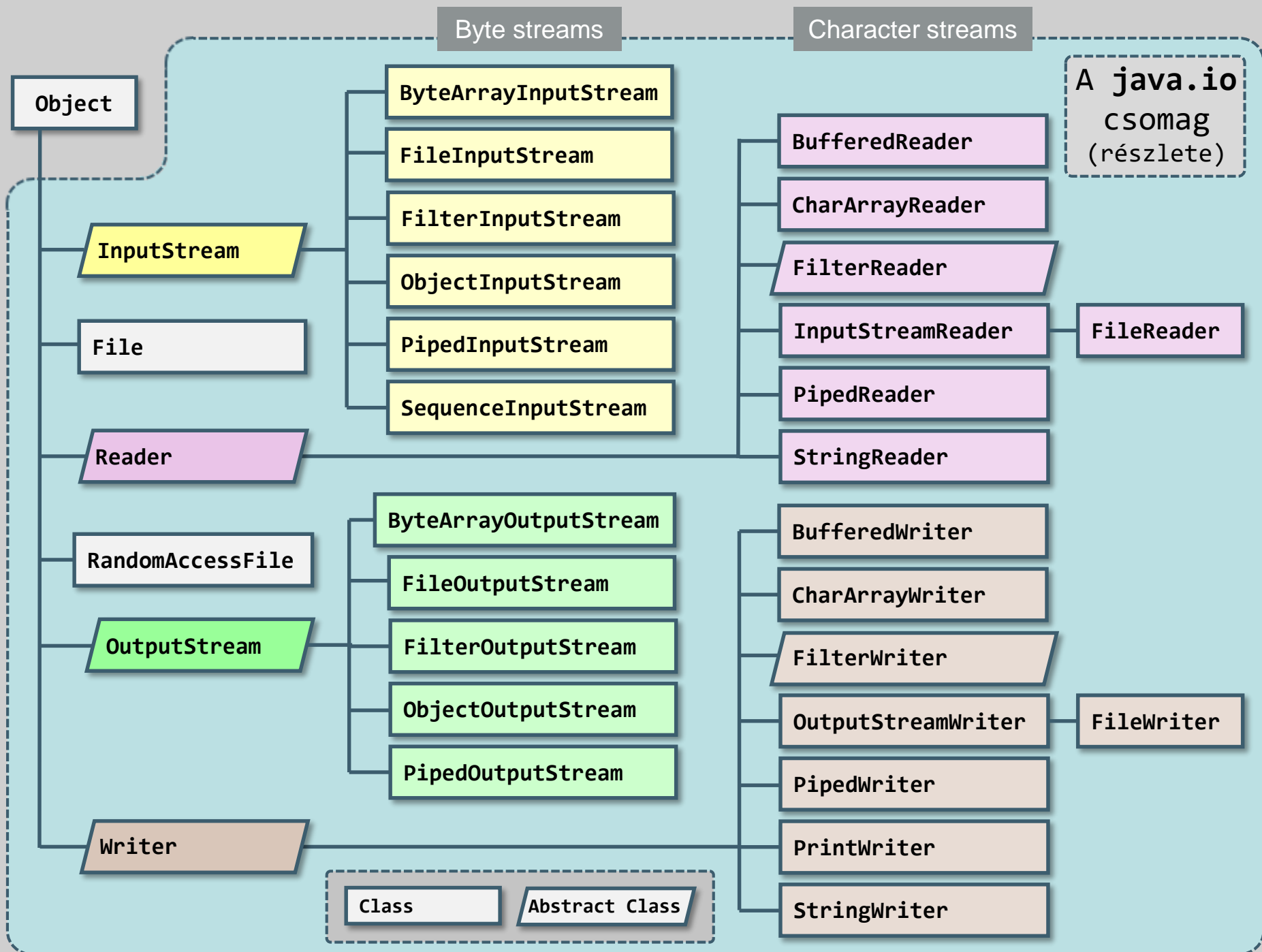
■ Csatornák csoportosítása:

- Irány szerint:
 - bemeneti csatorna
 - kimeneti csatorna
- Adattípus szerint:
 - byte szervezésű csatorna (bájtfolym)
 - karakter szervezésű csatorna (karakterfolym)
- Feladatuk alapján:
 - forrást illetve nyelőt meghatározó csatorna (pl. file, tömb, stb.)
 - szűrő csatorna:
 - A csatorna adatait módosítják
 - A csatorna funkcionalitását bővíti ki

Alap stream osztályok

| Típus - Irány | Bemeneti | Kimeneti |
|---------------|-------------|--------------|
| Bájt | InputStream | OutputStream |
| Karakter | Reader | Writer |

- Az ezekből származó osztályok ugyanígy végződnek.
- A két típus a csatorna által kezelt legkisebb adategység típusában különbözik:
 - Bájt csatorna: 8 bit
 - Karakter csatorna: 16 bit, (Unicode)
- A két osztálycsalád kezelő metódusainak elnevezése, használata megegyezik!



TESZT

Ön hülye?

☐

Igen

☐

Nem

Példakód – BufferedWriter/BufferedReader

```
1. import java.io.*;
   public class Fkezel {
       public static void main(String args[]) {
           String fnev = "TextFile.txt";
2.   BuWrite(fnev, "Hello Programmer!");
       BuWrAppend(fnev, "Good Work!");
       BuRead(fnev);
       }
       . . . Metódusok kódja
   }
```

String írás (olvasás) **szöveges fájlba.**

```
c:\Java8>java Fkezel
Data is written!
Data is appended!
Data from file:
Hello Programmer!
Good Work!
```

Példakód – BufferedWriter/BufferedReader

```
public static void BuWrite (String fileName, String data){  
1.   BufferedWriter writer = null;  
    try {  
2.       writer = new BufferedWriter(new FileWriter(fileName));  
3.       writer.write(data);  
       System.out.println("Data is written!");  
    } catch (IOException ioe) {  
       System.err.println (ioe);  
    } finally {  
        try {  
4.           writer.close();  
        } catch (IOException ioe) {  
           System.err.println (ioe);  
        }  
    }  
}
```

String írása (olvasása)
szöveges fájlba.

Példakód – BufferedWriter/BufferedReader

```
public static void BuWrAppend (String fileName, String data){  
    BufferedWriter writer = null;  
    try {  
        1. writer=new BufferedWriter(new FileWriter(fileName, true));  
        2. writer.newLine();  
        writer.write(data);  
        System.out.println("Data is appended!");  
    } catch (IOException ioe) {  
        System.err.println (ioe);  
    } finally {  
        try {  
            writer.close();  
        } catch (IOException ioe) {  
            System.err.println (ioe);  
        }  
    }  
}
```

String írása (olvasása)
szöveges fájlba.

Példakód – BufferedWriter/BufferedReader

```
public static void BuRead (String fileName){
```

```
1.   BufferedReader reader = null;
```

```
String s = "";
```

```
try {
```

```
2.   reader = new BufferedReader(new FileReader(fileName));
```

```
System.out.println("Data from file:");
```

```
3.   s = reader.readLine();
```

```
while(s != null) {
```

```
    System.out.println(s);
```

```
4.     s = reader.readLine();
```

```
}
```

```
} catch (IOException ioe) {
```

```
    System.err.println (ioe);
```

```
} finally {
```

```
    try {
```

```
5.        reader.close();
```

```
    } catch (IOException ioe) {
```

```
        System.err.println (ioe);
```

```
    }
```

```
}
```

```
}
```

String írása (olvasása)
szöveges fájlba.

```
c:\Java8>java Fkezel  
Data is written!  
Data is appended!  
Data from file:  
Hello Programmer!  
Good Work!
```


cafe java



Példakód – FileOutputStream/FileInputStream

```
1. import java.io.Serializable;
2. import java.util.Date;
3. import java.text.*;
   class Student implements Serializable{
       private String Name;
       private int Height;
4.   private Date Birthday;

       public Student (String n, int h, Date b) {
           Name = n;
           Height = h;
           Birthday = b;
       }

       public String toString() {
5.         SimpleDateFormat sdf = new SimpleDateFormat("yyyy.MM.dd");
6.         String s= sdf.format(Birthday);
           return "Name: "+Name+" Height: "+Height+" B.day: "+s;
       }
   }
```

Felhasználói osztály
kialakítása fájlkezelés
céljából.

Példakód – FileOutputStream/FileInputStream

```
1. import java.io.*;
import java.util.Date;
import java.text.*;

public class StudentPrg {
    private static Student st;
    private static int db = 3;
2. private static Student[] stA = new Student[10];
3. static SimpleDateFormat sdf = new SimpleDateFormat("yyyy.MM.dd");
    static Date d0=null, d1=null, d2=null;

    public static void main(String[] args) {
        try {
4.             d0 = sdf.parse("1999.12.21");
             d1 = sdf.parse("1995.04.09");
             d2 = sdf.parse("1998.07.14");
5.         } catch (ParseException pe) {System.out.println("Hiba: "+pe);}

        stA[0] = new Student("Béla", 188, d0);
        stA[1] = new Student("Enikő", 166, d1);
        stA[2] = new Student("Ödön", 179, d2);

        Kiir(db);
6.        Beolvas();
    } . . .
```

Felhasználói osztály írása
(olvasása) **bináris** fájlba.

Data from file:

Name: Béla Height: 188 B.day: 1999.12.21

Name: Enikő Height: 166 B.day: 1995.04.09

Name: Ödön Height: 179 B.day: 1998.07.14

Példakód – FileOutputStream/FileInputStream

`x=stA.lenght;`

```
1. public static void Kiir(int x) {  
    try {  
2.         FileOutputStream fos =  
                new FileOutputStream("Students.dat");  
3.         ObjectOutputStream oos =  
                new ObjectOutputStream(fos);  
4.         oos.writeInt(x);  
                for (int i = 0; i < x; i++) {  
5.                     oos.writeObject(stA[i]);  
                }  
6.         fos.close();  
                System.out.println("Data is written to file!");  
    } catch (IOException ioe) {  
        System.out.println("File write exception: "+ioe);  
    }  
}
```

Felhasználói osztály írása
(olvasása) bináris fájlba.

Student stA[];

Példakód – FileOutputStream/FileInputStream

Felhasználói osztály írása
(olvasása) **bináris** fájlba.

```
public static void Beolvas() {  
    try {  
1.      FileInputStream fis =  
           new FileInputStream("Students.dat");  
2.      ObjectInputStream ois = new ObjectInputStream(fis);  
3.      db = ois.readInt();  
4.      Object o = null;  
      System.out.println("Data from file:");  
      for (int i = 0; i < db; i++) {  
5.          o = ois.readObject();  
6.          st = (Student)o;  
          System.out.println(st);  
      }  
      fis.close();  
    } catch (Exception e) {  
        System.out.println("File write exception: "+e);  
    }  
}
```

Data from file:

Name: Béla Height: 188 B.day: 1999.12.21

Name: Enikő Height: 166 B.day: 1995.04.09

Name: Ödön Height: 179 B.day: 1998.07.14

Fájlrendszer kezelés

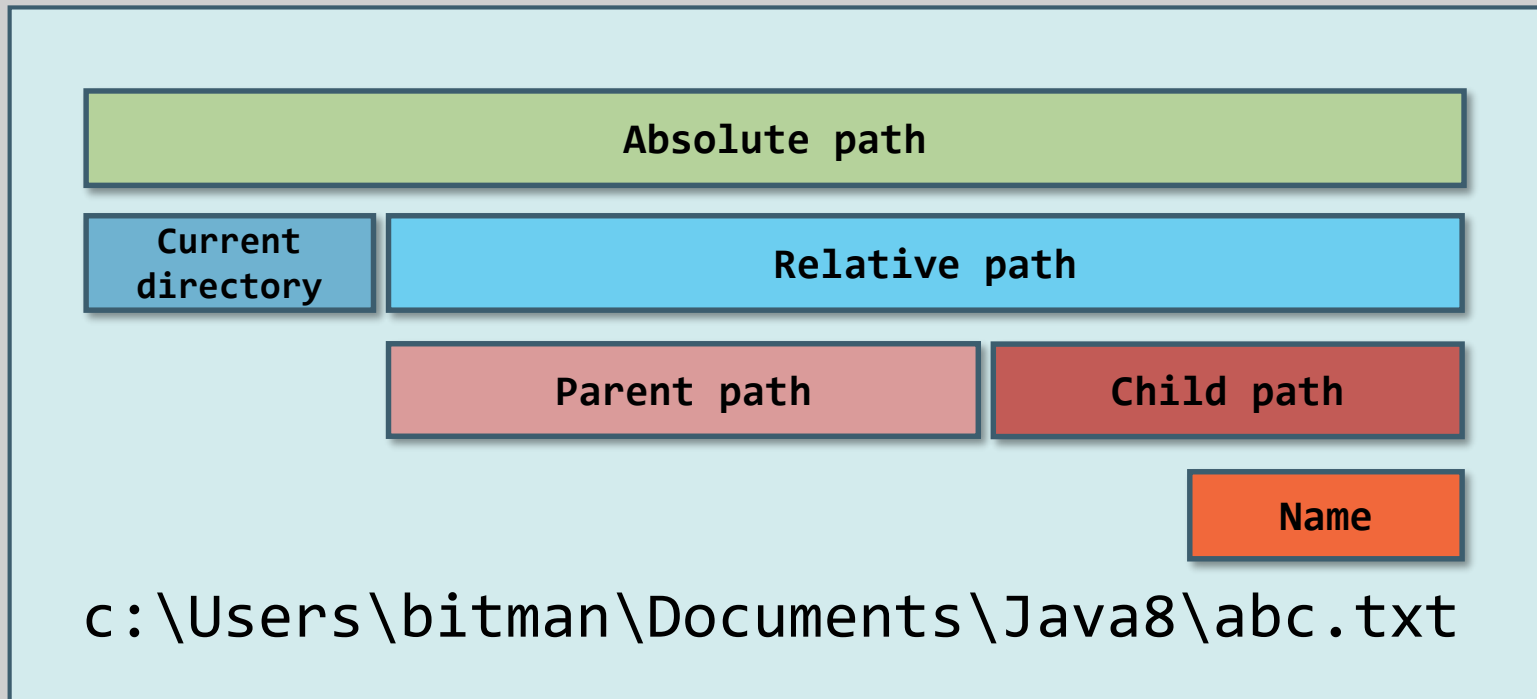
- A java.io csomagban vannak nem stream típusú fájlok kezelését szolgáló osztályok is, pl:
 - A `File` osztály egy fájlt reprezentál a helyi fájlrendszerben, segítségével fájl és könyvtárműveleteket végezhetünk
 - A `RandomAccessFile` osztály segítségével közvetlenül, direkt módon (nem csak sorosan) hozzáférhetünk egy fájl tartalmához
 - A `FilenameFilter` osztály segítségével szűrhetjük a megjelenített fájlnevek listáját

File osztály

- A `File` osztály egy fájlt (bejegyzést) reprezentál a helyi fájlrendszerben.
 - A fájl lehet könyvtár is, ami egy olyan fájl, amiben bejegyzések és egyéb információk lehetnek!
- A bejegyzést a rendszer az elérési útvonalával (path) azonosítja
- Konstruktorok
 - `File(String pathname);`
 - `File(String parent, String child);`
 - `File(File parent, String child);`

File osztály

■ Útvonalak értelmezése



Abszolút útvonal: mindig a root elemtől indul, és tartalmazza a könyvtárak listáját, a végén a fájl nevével.

Relatív útvonal: mindig kombinálni kell egy másik útvonallal, a fájl eléréséhez.

File osztály

■ A bejegyzés útvonal tulajdonságai :

- `boolean isAbsolute()` – abszolút a fájl útvonala?
- `String getAbsolutePath()` – abszolút útvonal lekérése
- `File getAbsoluteFile()` – fájl lekérése abszolút útvonallal
- `String getPath()` – relatív útvonal lekérése
- `String getParent()` – szülő útvonal lekérése
- `File getParentFile()` – szülő fájl lekérése
- `String getName()` – fájl neve

File osztály

■ A bejegyzés tulajdonságai:

- `boolean isDirectory()` – könyvtár?
- `boolean isFile()` – fájl?
- `long length()` – a fájl mérete
- `long lastModified()` – utolsó módosítás időpontja
- `boolean isHidden()` – rejtett?
- `boolean canRead()` – olvasható?
- `boolean canWrite()` – írható?
- `boolean exists()` – fizikailag létező bejegyzés?
- `boolean setReadOnly()` – csak olvasható beállítása
- `boolean setWritable(boolean)` – írható beállítása
- `boolean setReadable(boolean)` – olvasható beállítása

File osztály

■ A bejegyzések kezelése:

- `boolean mkdir()` – könyvtár létrehozása
- `boolean mkdirs()` – könyvtár létrehozása a nem létező szülőkönyvtárakkal együtt
- `boolean createNewFile()` throws `IOException` – üres fájl létrehozása (csak akkor jön létre, ha nem létezik)
- `boolean delete()` – bejegyzés törlése
- `void deleteOnExit()` – a VM leállásakor törli a fájlt
- `boolean renameTo(File dest)` – bejegyzés átnevezése
- `File[] listFiles()` – bejegyzésben lévő fájlok listája

java.nio csomag

- Új alternatíva a fájlok kezelésére
- A Java 7-es verziójában jelent meg
- Mindent tud, amit az io tudott
- Újdonságai:
 - Linkek kezelése
 - Metainformációk, jogosultságok kezelési lehetőségeinek bővítése
 - Több, egyszerűbb fájl kezelő parancs

A java.nio.file.attribute osztály

- Ez a csomag tartalmazza a fájlok attribútumait kezelő eljárásokat
 - A fájlok tulajdonságait, felhasználhatóságát, hatóköreit leíró jelölések az attribútumok.
 - Az, hogy milyen attribútumokat vehetnek fel a fájlok, az adott fájlrendszerrel függ.
 - Az attribútumok attribútum csoportokba vannak szervezve:
 - Basic,
 - Dos,
 - FileOwner,
 - Posix,
 - Acl,
 - UserDefined

9

10



Java
JDBC

- Az **SQLite** önálló, fájl alapú, kisméretű, C forrású programkönyvtárként megvalósított ACID-kompatibilis relációs adatbázis-kezelő rendszer, illetve adatbázismotor.
- A szoftvert D. Richard Hipp tervezte és alkotta meg.
- Az SQLite forráskódja nyílt, közkincsnek számít.
- Egyre több népszerű szoftver használja:
 - Mozilla Firefox
 - Adobe Reader, Acrobat, Photoshop
 - Airbus
 - Apple
 - Facebook
 - Microsoft
 - Google
 - PHP

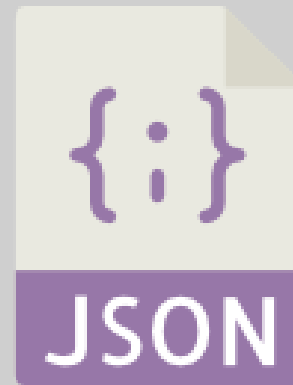




10

1

1



11

**Vége. Jöhet a
gyakorlat.**

Feladat

- Készítsünk egy adatkezelő alkalmazást, mely egy dolgozónak a következő adatait kezeli:
 - Kód
 - Név
 - Születési idő
 - Lakcím
 - Fizetés
- Az alkalmazás legyen alkalmas:
 - az adatok felvitelére,
 - listázására,
 - módosítására,
 - törlésére.

Feladat

- Az adatokat grafikus felületű alkalmazással fogjuk kezelni
- Az adatok megjelenítése egy táblázat segítségével fog történni, ezért az adatokat egy táblamodellben fogjuk tárolni.
- Mielőtt a grafikus felületet elkészítenénk, hozzunk létre egy adatokat tartalmazó szöveges fájlt, olvassuk be, és jelenítsük meg a szöveges képernyőn az adatokat.
 - Ez azért lesz nagy segítség, mert így tudjuk, hogy vannak a fájlban adatok, elérjük, és be tudjuk olvasni az adatokat.
 - Egyszerűbb úgy megírni a táblázatos adatmegjelenítést, hogy biztosak vagyunk abban, hogy az adatok léteznek, és be tudjuk azokat olvasni.

Feladat

- A szöveges fájl egy csv típusú fájl lesz, mely pontosvesszővel elválasztva tartalmazza az egyes adatsorokat, de minden adatsor külön sorban van.
- Készítsünk egy Java projektet az Eclipseben, legyen a neve: [Adatkezelő_ozd](#)
- Nyissuk meg a projekt munkakönyvtárát, és a gyökerébe Jegyzettömbbel hozzunk létre egy [adatok.csv](#) nevű fájlt

Feladat

■ A projekt munkakönyvtárának megnyitása:

The screenshot illustrates the steps to open a project's working set in Eclipse IDE. The steps are numbered 1 through 6:

- 1**: Right-click on the project 'Adatkezes_ozd' in the Package Explorer.
- 2**: Select 'Properties' from the context menu (labeled 'Alt+Enter').
- 3**: In the 'Properties for Adatkezes_ozd' dialog, select 'Resource' from the left-hand list.
- 4**: Click the 'Show In System Explorer' button in the 'Resource' tab.
- 5**: In the System Explorer, navigate to 'eclipse-workspace > Adatkezes_ozd'.
- 6**: Click on the 'adatok.csv' file in the project view.

The 'Properties for Adatkezes_ozd' dialog shows the following details:

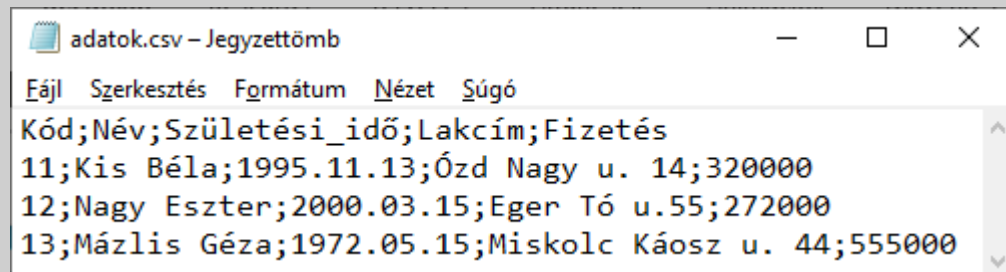
- Resource**
- Path:** /Adatkezes_ozd
- Type:** Project
- Location:** C:\Users\bitman\eclipse-workspace\Adatkezes_ozd
- Last modified:** 2020. március 1. 14:35:45
- Text file encoding:** ☒ Inherited from container (Cp1250)
- Other:** Cp1250
- Store the encoding of derived resources separately:** ☐
- New text file line delimiter:** ☒ Inherited from container (Windows)
- Other:** Windows

The System Explorer shows the project structure:

- bin
- src
- .classpath
- .project
- adatok.csv

The 'BITMAN' logo is visible in the bottom right corner.

adatok.csv



adatok.csv - Jegyzetkönyv

Fájl Szerkesztés Formátum Nézet Súgó

```
Kód;Név;Születési_idő;Lakcím;Fizetés
11;Kis Béla;1995.11.13;Ózd Nagy u. 14;320000
12;Nagy Eszter;2000.03.15;Eger Tó u.55;272000
13;Mázlis Géza;1972.05.15;Miskolc Káosz u. 44;555000
```

```
Kód;Név;Születési_idő;Lakcím;Fizetés ↵
11;Kis Béla;1995.11.13;Ózd Nagy u. 14;320000 ↵
12;Nagy Eszter;2000.03.15;Eger Tó u.55;272000 ↵
13;Mázlis Géza;1972.05.15;Miskolc Káosz u. 44;555000 ↵
```

Fájl beolvasása

- Készítsünk egy metódust, mely beolvassa, és a képernyőre kiírja a fájl tartalmát:

- Eclipse-ben:

- ① – Fájl\New\Class: neve legyen Program
+ `public static void main(String[] args)`
- ① – Fájl\New\Class: neve legyen FileManager
 - Írjuk meg a `CsvReader` metódust

CsvReader

| Kód | Név | Születési_idő | Lakcím | Fizetés |
|-----|-------------|---------------|---------------------|---------|
| 11 | Kis Béla | 1995.11.13 | Ózd Nagy u. 14 | 320000 |
| 12 | Nagy Eszter | 2000.03.15 | Eger Tó u.55 | 272000 |
| 13 | Mázlis Géza | 1972.05.15 | Miskolc Káosz u. 44 | 555000 |

```
public class FileManager {  
  
    public static void CsvReader() {  
        String x=" - ";  
        try {  
            BufferedReader in = new BufferedReader(new FileReader("adatok.csv"));  
            String s=in.readLine();    //=== mezőnevek az első sorból  
            s=in.readLine();           //=== 1.adatsor  
            while(s!=null) {  
                String[] st = s.split(";");  
                System.out.println(st[0]+x+st[1]+x+st[2]+x+st[3]+x+st[4]);  
                s=in.readLine();  
            }  
            in.close();  
        } catch (IOException ioe) {  
            System.out.println("CsvReader: "+ioe.getMessage());  
        }  
    }  
}
```

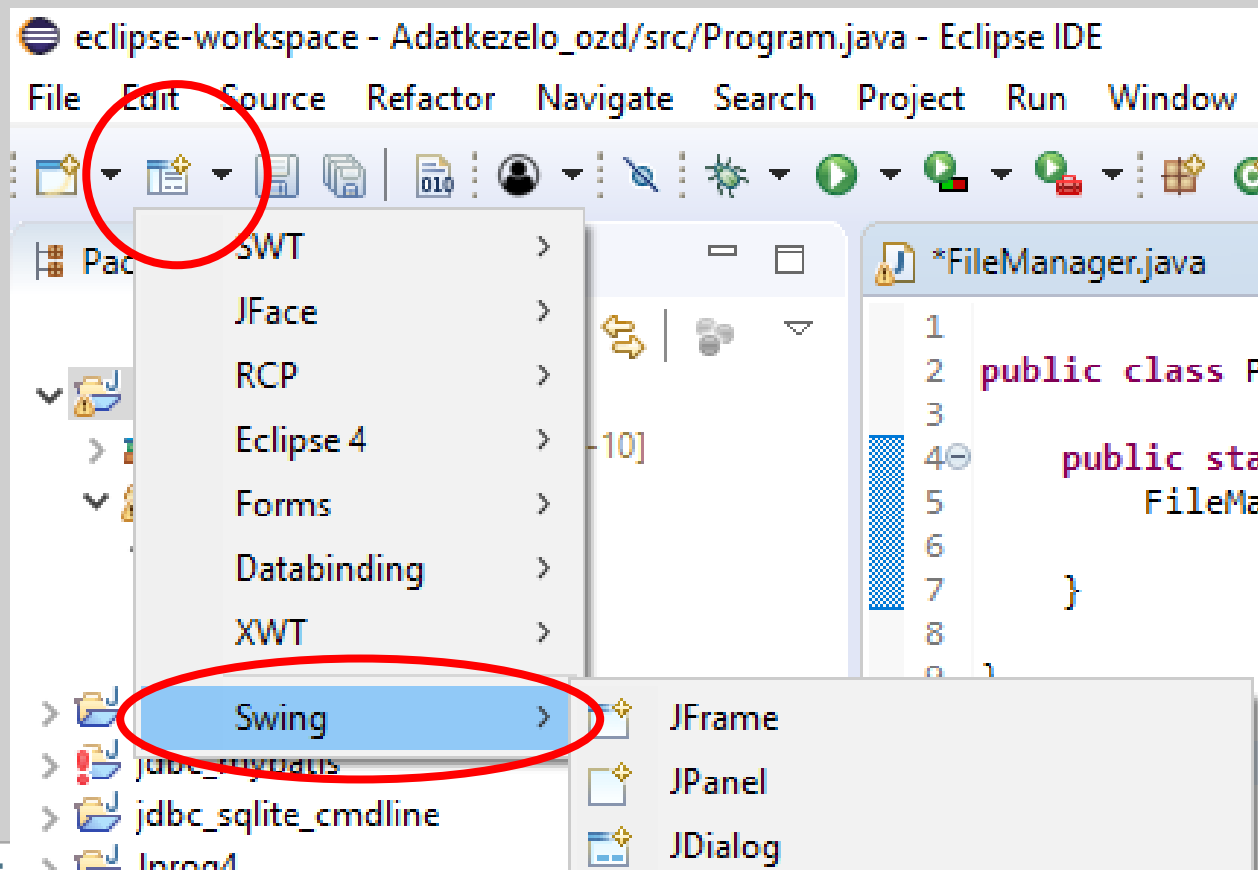
Program

```
public class Program {  
  
    public static void main(String[] args) {  
        FileManager.CsvReader();  
    }  
  
}
```

```
11 - Kis Béla - 1995.11.13 - Ózd Nagy u. 14 - 320000  
12 - Nagy Eszter - 2000.03.15 - Eger Tó u.55 - 272000  
13 - Máztis Géza - 1972.05.15 - Miskolc Káosz u. 44 - 555000
```

Jöhet a grafikus felület

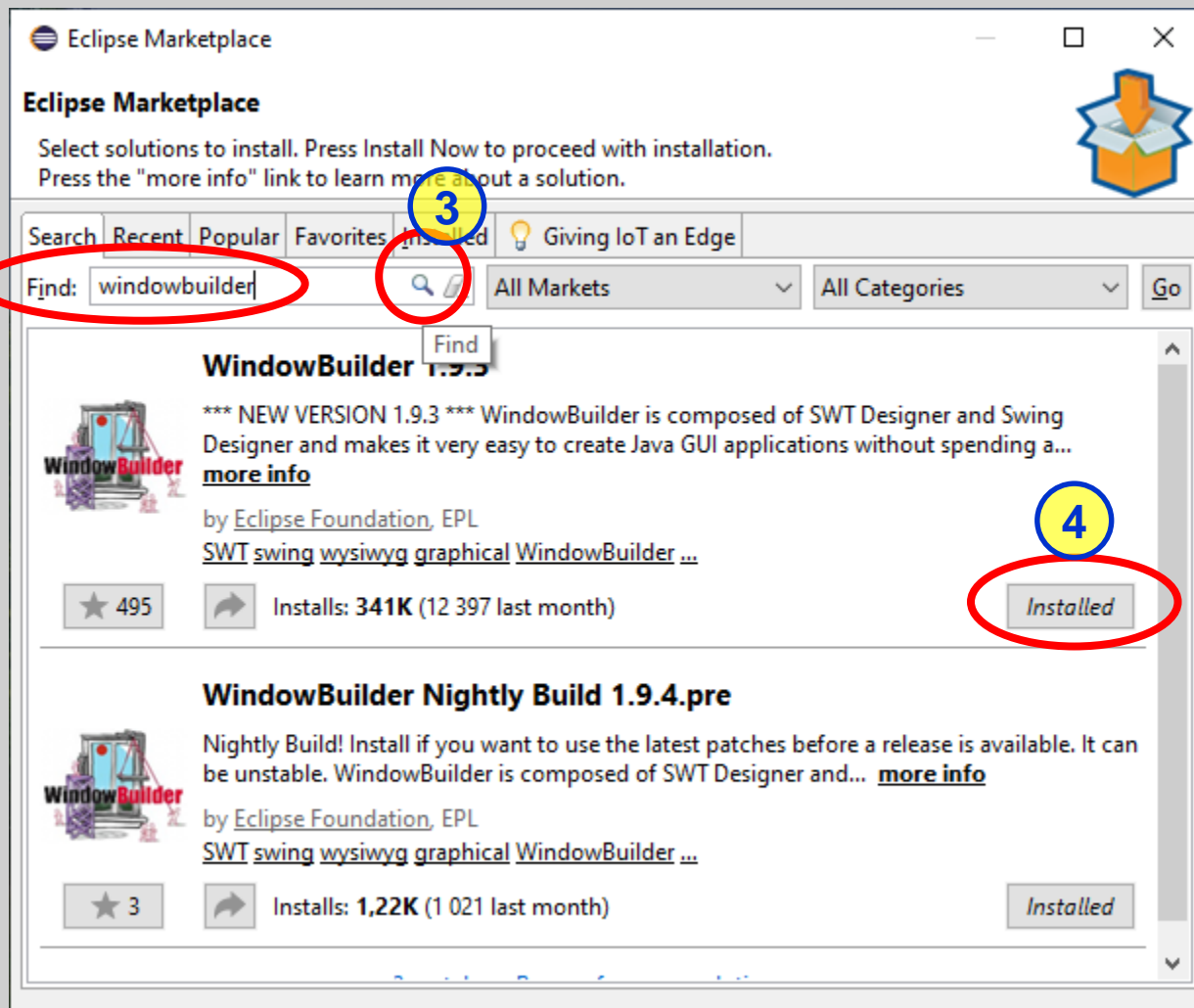
- Ellenőrizzük, hogy telepítve van-e az Eclipse-ben a grafikus tervezőfelület, melynek neve: [WindowBuilder](#).
- Ha igen, akkor látható a következő ikon az Eclipse-ben:



Ha nem lenne a WindowBuilder telepítve az Eclipse-ben:

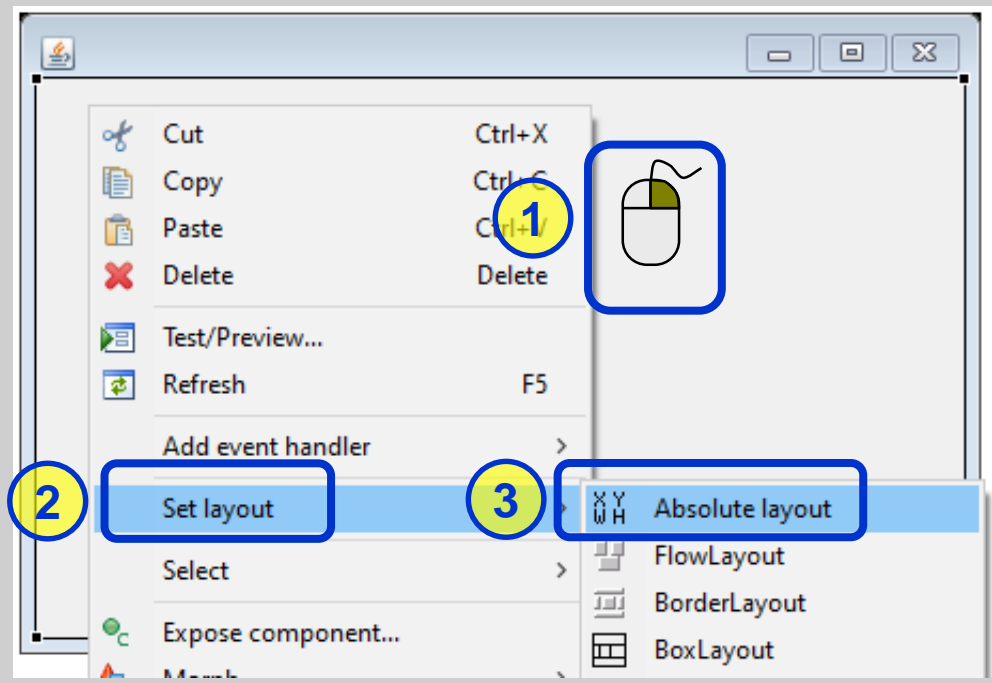
1

■ Help\Eclipse Marketplace



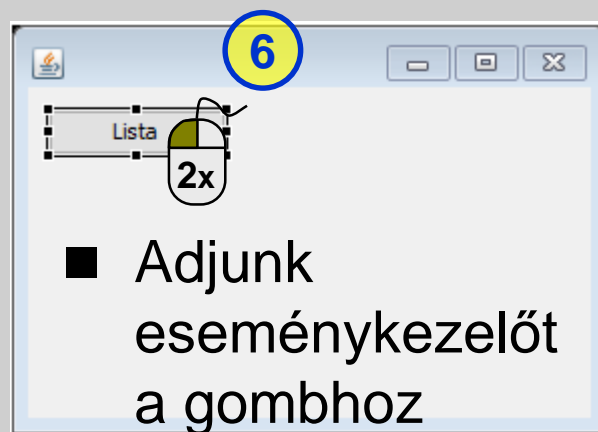
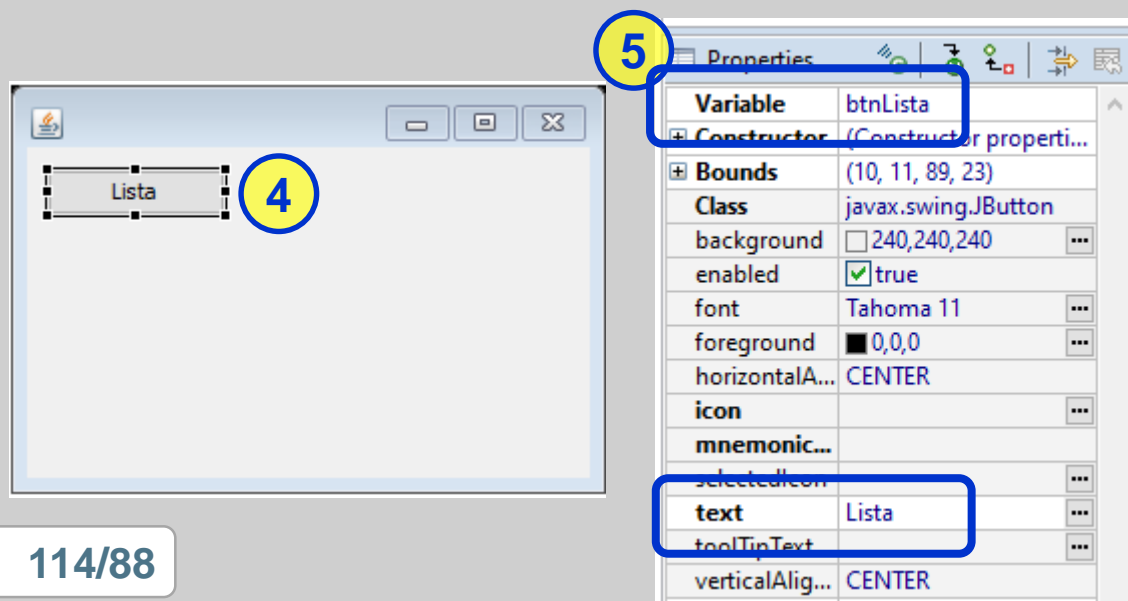
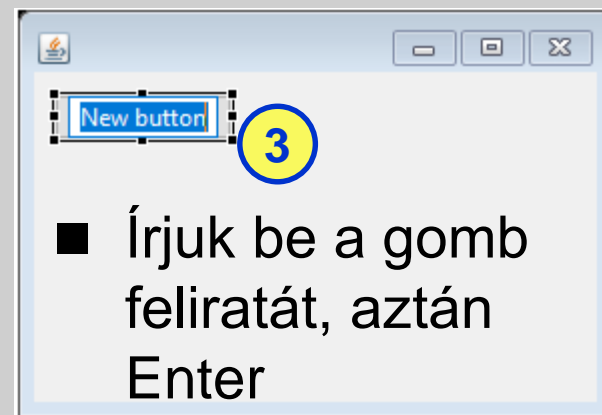
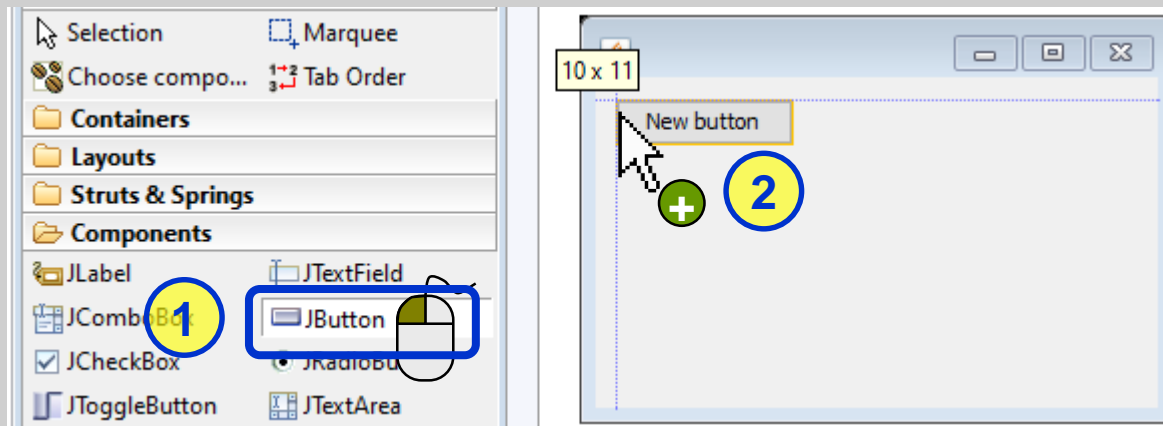
A grafikus program

- Először töröljük le a **Program** nevű osztályt:
 - Jobb klikk az osztály nevére\Delete
- Hozzuk létre a projektünkhöz egy új grafikus ablakot:
 - 2.ikon**Swing****JFrame**: neve legyen **Program**
- Menjünk át **Design** nézetre
- Kattintsunk jobb gombbal az ablak közepébe, és kapcsoljuk ki a tartalom elrendezőt:



A grafikus program

■ Tegyük egy nyomógombot a panelre

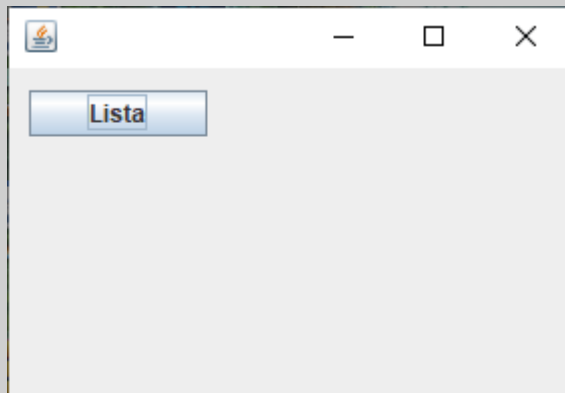


A grafikus program

- Írjuk be az esemény kódját:

```
JBUTTON btnLista = new JButton("Lista");  
btnLista.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        FileManager.CsvReader();  
    }  
});
```

- Indítsuk el a programot, és próbáljuk ki a Listázást



```
11 - Kis Béla - 1995.11.13 - Ózd Nagy u. 14 - 320000  
12 - Nagy Eszter - 2000.03.15 - Eger Tó u.55 - 272000  
13 - Máztis Géza - 1972.05.15 - Miskolc Káosz u. 44 - 555000
```

Ha elakadtál küldj ímélt a
help@help.com címre!



Vagy tedd fel a kezed!

Lista panel kialakítás – tábla modell létrehozása



The screenshot shows a Java Swing window titled "Dolgozók listája" (Employees list). It contains a table with 6 columns: Jel (checkbox), Kód, Név, Születő, Lakóhely, and Fizetés. There are three rows of data. Below the table is a "Bezár" (Close) button.

| Jel | Kód | Név | Születő | Lakóhely | Fizetés |
|--------------------------|-----|-------------|------------|---------------------|---------|
| <input type="checkbox"/> | 11 | Kis Béla | 1995.11.13 | Ózd Nagy u. 14 | 320000 |
| <input type="checkbox"/> | 12 | Nagy Eszter | 2000.03.15 | Eger Tó u.55 | 272000 |
| <input type="checkbox"/> | 13 | Mázlis Géza | 1972.05.15 | Miskolc Káosz u. 44 | 555000 |

- A táblázat kezelése egy táblamoddellel kezdődik.
- A táblamodell egy formázható tárolószervezet, ebben lesznek a táblázat adatai, ez tartalmazza az egyes oszlopok típusát és szerkeszthetőségét.
- A modellhez sorokban lehet hozzáadni adatokat: `tm.addRow(...)`
- A modelltől mezőnként lehet lekérdezni az adatokat: `tm.getValueAt(row, column)`
- A táblamodell a [DefaultTableModel](#) leszármaztatásával jön létre, konstruktorában kapja meg a mezőneveket, és az oszlopok számát.

Lista panel kialakítás – tábla modell létrehozása



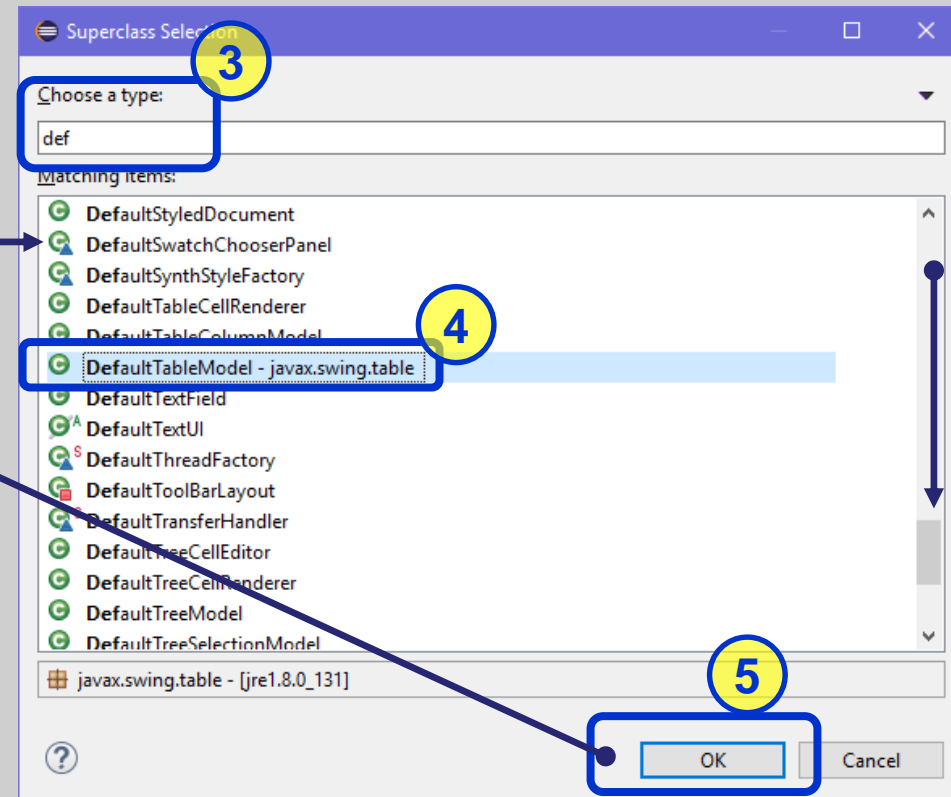
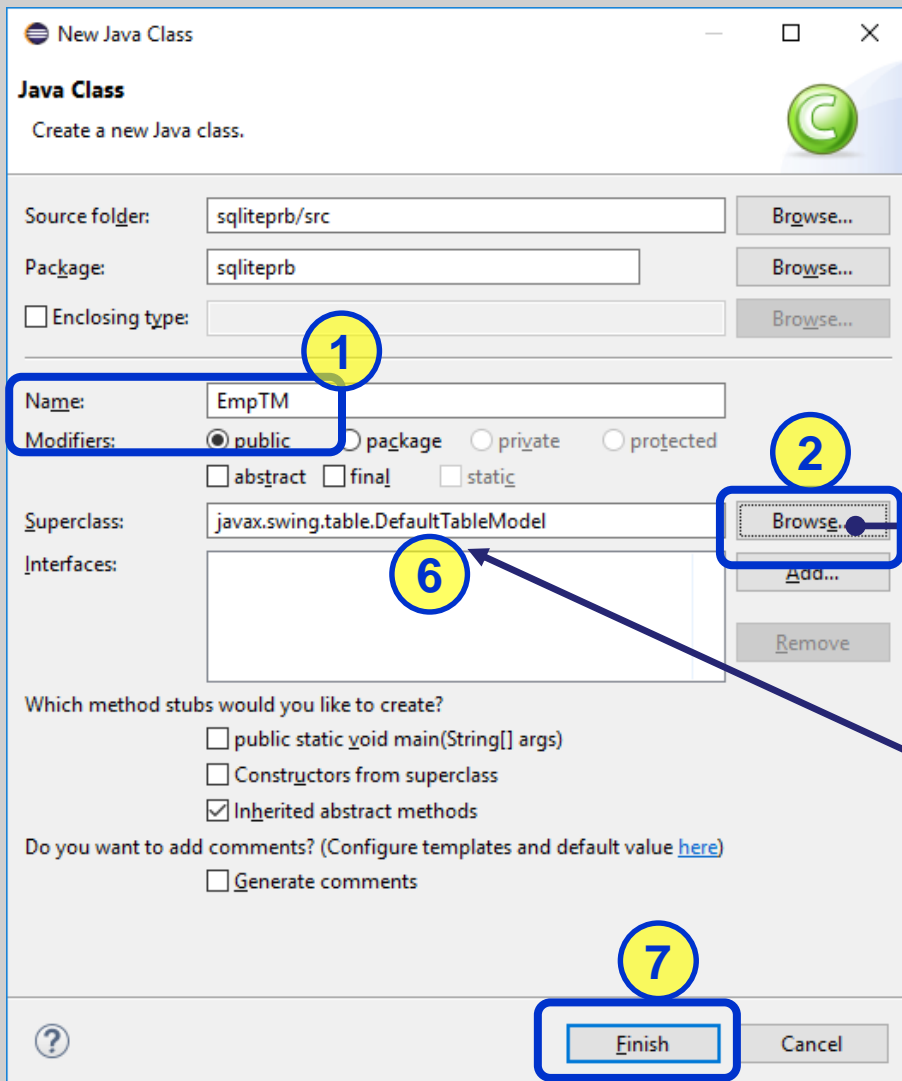
| Jel | Kód | Név | Születő | Lakóhely | Fizetés |
|--------------------------|-----|-------------|------------|---------------------|---------|
| <input type="checkbox"/> | 11 | Kis Béla | 1995.11.13 | Ózd Nagy u. 14 | 320000 |
| <input type="checkbox"/> | 12 | Nagy Eszter | 2000.03.15 | Eger Tó u.55 | 272000 |
| <input type="checkbox"/> | 13 | Mázlis Géza | 1972.05.15 | Miskolc Káosz u. 44 | 555000 |

Programlogika:

1. Egy modellt használnánk csak a programban az egyszerűség miatt
2. A módosításnál és a törlésnél ki kell jelölni egy rekordot, így szükséges egy jelölhető (logikai típusú) mező a sor elejére
3. A mezők típusa rendre:
logikai, egész, szöveg, szöveg, szöveg, egész
3. Csak a legelső (logikai) mezőnek kell szerkeszthetőnek lennie (a kijelölhetőség miatt!)

Lista panel kialakítás – tábla modell létrehozása

■ File \ New \ Class - Neve legyen: EmpTM



Lista panel kialakítás – tábla modell létrehozása

■ A létrejött EmpTM osztály

```
import javax.swing.table.DefaultTableModel;

public class EmpTM extends DefaultTableModel {

}
```

- A táblamodellben kötelező létrehozni a következő elemeket:
 - **Konstruktor**: mely a paraméterként megadott oszlopnevekkel és sorok számával létrehozza a példányt
 - Az **isCellEditable** metódus, melyben megadjuk, hogy mely oszlopok lesznek szerkeszthetők, és melyek nem
 - A **getColumnClass** metódus, melyben megadjuk, hogy melyik oszlop milyen adattípusú lesz

Lista panel kialakítás – tábla modell létrehozása

```
import javax.swing.table.DefaultTableModel;

public class EmpTM extends DefaultTableModel {
    public EmpTM (Object fildNames[], int rows){
        super(fildNames, rows);
    }

    public boolean isCellEditable(int row, int col) {
        if (col == 0) {return true;}
        return false;
    }

    public Class<?> getColumnClass(int index){
        if (index == 0) return(Boolean.class);
        else if (index==1 || index==5) return(Integer.class);
        return(String.class);
    }
}
```

Zöld: kötelező rész, minden táblamodellben egyforma!

Konstruktor: megkapja a mezők nevét és a sorok számát.

Szerkeszthetőség: a 0. oszlop minden sora szerkeszthető, a többi cella nem!

Oszlopok típusa: a 0. oszlop **logikai**, az 1. és az 5. **egész**, a többi **szöveges**!

Listázás

1. Panel létrehozása:

New\Swing\JDialog – Neve: EmpList

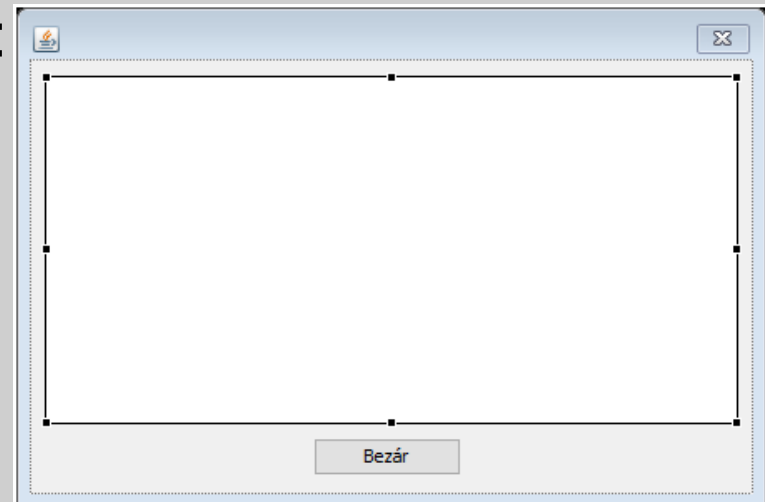
- A *Generate JDialog with OK and Cancel buttons* opciót kapcsoljuk ki!

2. Panelen az elrendezés menedzser kikapcsolása:

Jobb klikk a panel közepére\Set layout\Absolute layout

3. Készítsük el e Bezár (btnBezar) gombot, aztán adjunk hozzá eseménykezelőt.

4. Adjunk egy JTable-t a panelhez:



Listázás

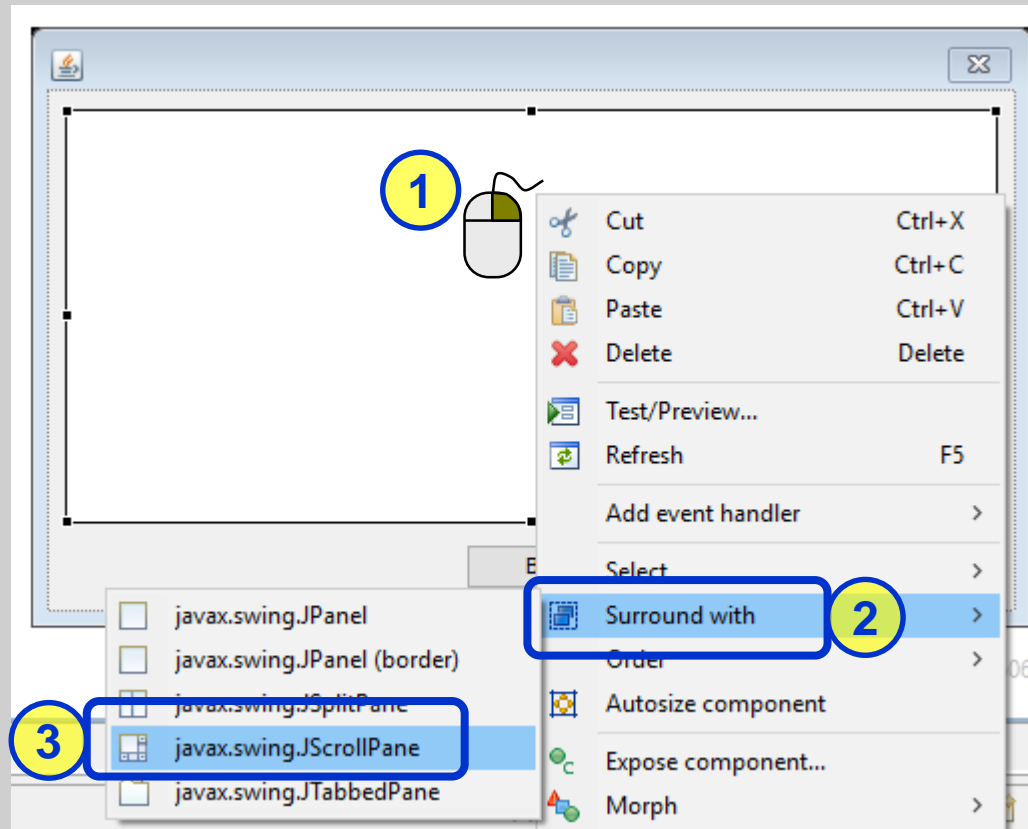
■ Adjunk egy.JTable-t a panelhez:

The diagram illustrates the process of adding a `JTable` component to a panel in a GUI builder. It consists of three main parts:

- Component Selection:** A yellow circle with the number '1' highlights the `JTable` component in the 'Components' list on the left. The list includes various Java Swing components like `JLabel`, `JTextF...`, `JCom...`, `JButton`, `JChec...`, `JRadi...`, `JTogg...`, `JText...`, `JForm...`, `JPass...`, `JTextP...`, `JEdito...`, `JSpin...`, `JList`, `JTree`, `JProgr...`, and `JScroll...`.
- Dragging and Resizing:** A yellow circle with the number '2' shows a mouse cursor dragging the `JTable` component (labeled '10 x 11') from the components list onto a panel. A yellow circle with the number '3' shows the mouse cursor at the bottom-right corner of the table, which is now labeled '414 x 204'. A blue arrow indicates the movement from step 2 to step 3. A 'Bezár' (Close) button is visible at the bottom of the panel.
- Final Result:** A yellow circle with the number '4' shows the final state where the `JTable` component is successfully added to the panel. The table is now a large white rectangle within the panel, and the 'Bezár' button remains at the bottom.

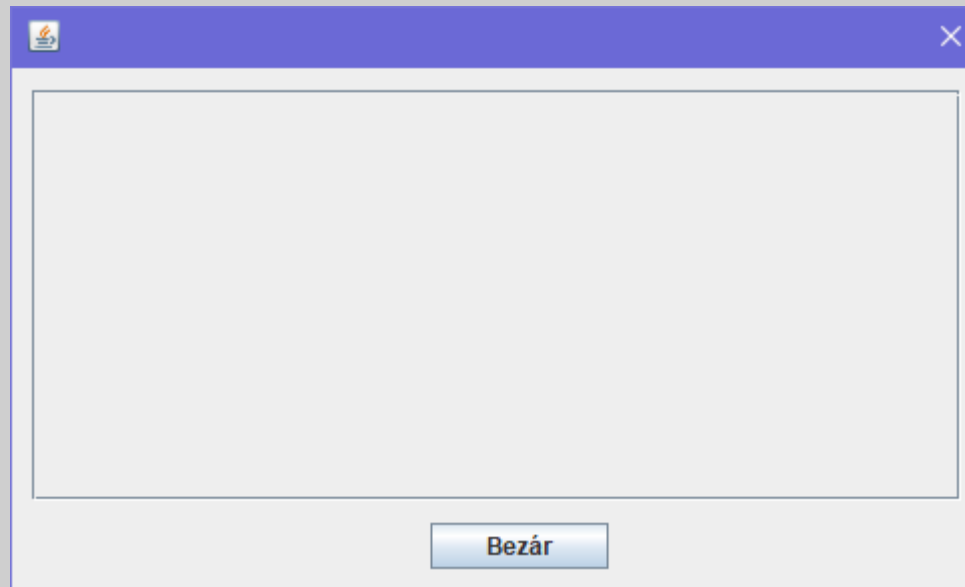
Lista panel kialakítás – a panel létrehozása

- Adjunk a `JTable`-hez egy `JScrollPane`-t (görgetősávok)



Lista panel kialakítás – a panel létrehozása

- Futtassuk a kódot, nézzük meg a panelt!



Az EmpList panel átalakítása

■ Kódbűvölés következik!

1 Az osztály elején hozzunk létre egy **EmpTM** példányt!

2 A kódból töröljük a main függvényt!


Nem lesz önállóan futtatható a panel, ezért **nem kell** bele a **main** függvény!

```
public class EmpList extends JDialog {  
  
    private final JPanel contentPanel = new JPanel();  
    private JTable table;  
    private EmpTM etm;  
  
    /**  
     * Launch the application.  
     */  
    public static void main(String[] args) {  
        try {  
            EmpList dialog = new EmpList();  
            dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);  
            dialog.setVisible(true);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
  
    /**  
     * Create the dialog.  
     */  
    public EmpList() {  
        setBounds(100, 100, 450, 300);  
        getContentPane().setLayout(new BorderLayout());  
    }  
}
```

Az EmpList panel átalakítása

- Módosítsuk a JTable példányosításának kódját:

```
public EmpList() {  
    setBounds(100, 100, 450, 300);  
    getContentPane().setLayout(new BorderLayout());  
    contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));  
    getContentPane().add(contentPanel, BorderLayout.CENTER);  
    contentPanel.setLayout(null);  
    {  
        JButton btnBezr = new JButton("Bez\u00E1r");  
        btnBezr.setBounds(172, 227, 89, 23);  
        contentPanel.add(btnBezr);  
    }  
  
    JScrollPane scrollPane = new JScrollPane();  
    scrollPane.setBounds(10, 11, 414, 204);  
    contentPanel.add(scrollPane);  
  
    table = new JTable(etm);  
    scrollPane.setViewportView(table);  
}
```



`table = new JTable(etm);`

A táblázatban az **etm**
modell fog megjelenni

Az EmpList panel átalakítása

Alakítsuk át a
konstruktor elejét:

```
public EmpList(JFrame f, EmpTM betm) {  
    super(f, "Dolgozók listája", true);  
    etm = betm;  
}
```

Tulajdonos, ablak címe, modális jelző

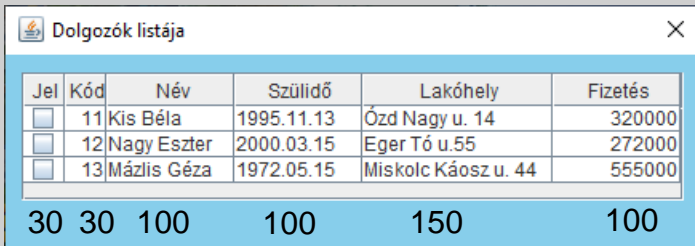
Modális ablak lesz, melyet be
kell zárni ahhoz, hogy az alatta
lévő ablak újra aktív legyen!

```
public class EmpList extends JDialog {  
  
    private final JPanel contentPanel = new JPanel();  
    private JTable table;  
    private EmpTM etm;  
  
    public EmpList(JFrame f, EmpTM betm) {  
        super(f, "Dolgozók listája", true);  
        etm = betm;  
  
        setBounds(100, 100, 450, 300);  
        getContentPane().setLayout(new BorderLayout());  
        contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));  
        getContentPane().add(contentPanel, BorderLayout.CENTER);  
    }  
}
```

Az EmpList panel átalakítása

A konstruktor kódjának a legvégére írjuk be a következőket:

```
TableColumn tc = null;
for (int i = 0; i < 6; i++) {
    tc = table.getColumnModel().getColumn(i);
    if (i==0 || i==1) tc.setPreferredWidth(30);
    else if (i==4) tc.setPreferredWidth(150);
    else {tc.setPreferredWidth(100);}
}
```



The screenshot shows a window titled "Dolgozók listája" containing a table with 6 columns: Jel, Kód, Név, Szülidő, Lakóhely, and Fizetés. The table has 3 rows of data. Below the table, the preferred widths for each column are listed: 30, 30, 100, 100, 150, and 100.

| Jel | Kód | Név | Szülidő | Lakóhely | Fizetés |
|--------------------------|-----|-------------|------------|---------------------|---------|
| <input type="checkbox"/> | 11 | Kis Béla | 1995.11.13 | Ózd Nagy u. 14 | 320000 |
| <input type="checkbox"/> | 12 | Nagy Eszter | 2000.03.15 | Eger Tó u.55 | 272000 |
| <input type="checkbox"/> | 13 | Mázlis Géza | 1972.05.15 | Miskolc Káosz u. 44 | 555000 |

30 30 100 100 150 100

Oszlopszélesség megadása: a teljes szélességet felosztja az itt megadott értékek arányában!

Az EmpList panel átalakítása

A konstruktor kódjának a legvégére (az előző kód alá) írjuk be a következőket:

```
table.setAutoCreateRowSorter(true);  
TableRowSorter<EmpTM> trs =  
    (TableRowSorter<EmpTM>)table.getRowSorter();  
trs.setSorttable(0, false);
```

AutoSorter bekapcsolása

A 0. oszlop rendezhetőségének
letiltása


Az EmpList panel átalakítása

- A konstruktor kódjának a vége:

```
JScrollPane scrollPane = new JScrollPane();
scrollPane.setBounds(10, 11, 414, 204);
contentPanel.add(scrollPane);

table = new JTable(etm);
scrollPane.setViewportView(table);

TableColumn tc = null;
for (int i = 0; i < 6; i++) {
    tc = table.getColumnModel().getColumn(i);
    if (i==0 || i==1 || i==5) tc.setPreferredWidth(30);
    else {tc.setPreferredWidth(100);}
}
table.setAutoCreateRowSorter(true);
TableRowSorter<EmpTM> trs =
    (TableRowSorter<EmpTM>)table.getRowSorter();
trs.setSorttable(0, false);
}
```



Az EmpList panel átalakítása – a panel bezárása

- Kattintsunk duplán a **Bezár** gombon **Dizájn** nézetben, így az Eclipse hozzáad egy eseménykezelőt a gombhoz, és megnyitja a kódot!

Írjuk be a **Bezár** gomb kódját:

```
JButton btnBezar = new JButton("Bez\u00E1r");  
btnBezar.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        dispose();  
    }  
});
```

dispose(): eltűnik a képernyőről a panel, de a referenciái megmaradnak.

A Lista panel megjelenítése (a Program-ban)

■ Kódbűvölés következik!

Az osztály elejére:

```
private EmpTM etm;
```

A **Lista** gomb kódja:

```
btnLista.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        FileManager.CsvReader();  
        EmpList el = new EmpList(Program.this, etm);  
        el.setVisible(true);  
    }  
});
```

Megjelenítés

Példányosítás

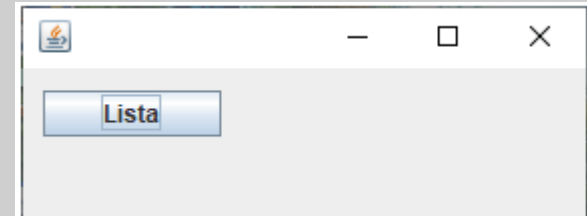
A konstruktor legvégére:

Mezőnevek megadása

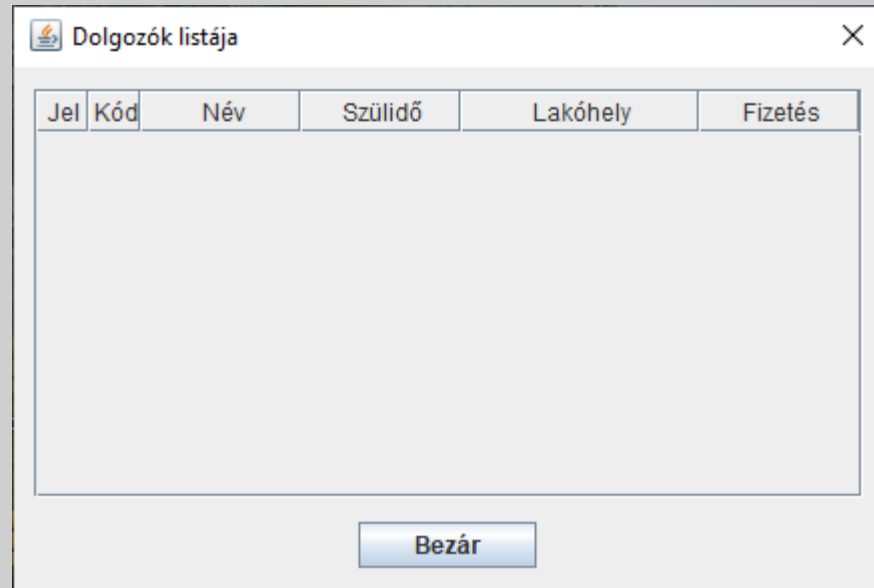
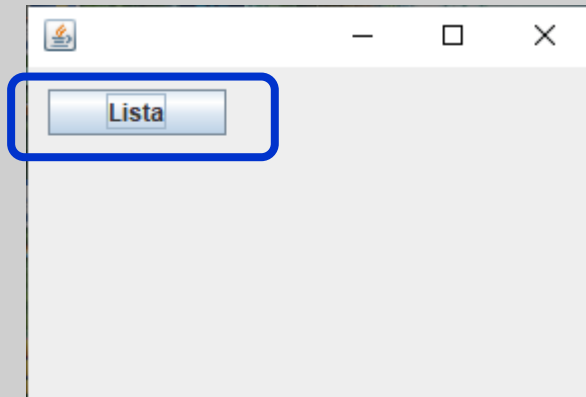
```
Object emptmn[] = {"Jel", "Kód", "Név", "Szüldő", "Lakóhely", "Fizetés"};  
etm = new EmpTM(emptmn, 0);
```

etm példányosítása:

mezőnevekkel, 0 darab sorral



A Lista panel kipróbálása (az EmpProgramban)



A CsvReader kódjának módosítása

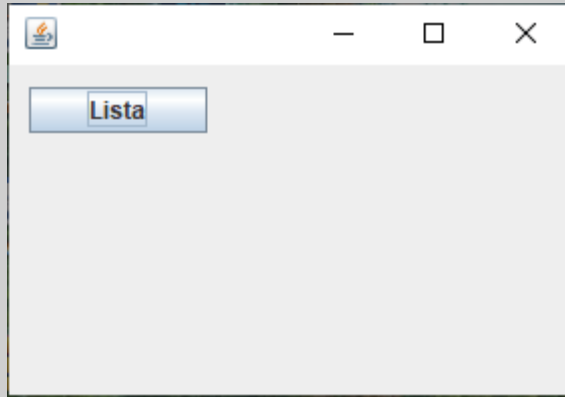
```
public class FileManager {  
      
    public static EmpTM CsvReader() {  
         Object emptmn[] = {"Jel", "Kód", "Név", "Szülidő", "Lakóhely", "Fizetés"};  
         EmpTM etm = new EmpTM(emptmn, 0);  
        try {  
            BufferedReader in = new BufferedReader(new FileReader("adatok.csv"));  
            String s=in.readLine();  
            s=in.readLine();  
            while(s!=null) {  
                 String[] st = s.split(";");  
                etm.addRow(new Object[]{false, st[0], st[1], st[2], st[3], st[4]});  
                s=in.readLine();  
            }  
            in.close();  
        } catch (IOException ioe) {  
            System.out.println("CsvReader: "+ioe.getMessage());  
        }  
         return etm;  
    }  
}
```

A Lista gomb kódjának módosítása

```

JButton btnLista = new JButton("Lista");
btnLista.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        ➡ etm = FileManager.CsvReader();
        EmpList el = new EmpList(Program.this, etm);
        el.setVisible(true);
    }
});
```

A program kipróbálása



Ha elakadtál
küldj ímélt a
help@help.com
címrre!



Vagy tedd
fel a
táblád!



Question
van?

VÉGE

