Objektumorientált programozás

Genericitás és IO

Goldschmidt Balázs

balage@iit.bme.hu

Genericitás

Mi történik az *ArrayList*-nél?

```
Collection<Integer> l2 = new ArrayList<Integer>();
Collection<Integer> l3 = new ArrayList<Integer>();
```

Mit változtat az <*Integer*> ?

```
for (int i = 0; i < 100; i++) {
      l2.add(new Integer(i*2));
      l3.add(new Integer(i*3));
}
Collection<Integer> l6 = new ArrayList<Integer>();
l6.addAll(l2); // l2 tartalma l6-ba
l6.retainAll(l3); // csak a 3-mal oszthatók maradnak
```

Csináljunk *Stack*-et *double*-höz!

```
// írassunk ki visszafele
Stack s =
       new Stack();

s.push(1.0);
s.push(2.71);
s.push(3.14);

while (s.size()>0) {
  Double d = s.pop();
  System.out.println(d);
}
```

```
public class Stack {
  ArrayList<Double> l;
  public Stack() {
    l = new ArrayList<Double>();
  }
  public void push(Double d) {
    l.add(d);
  }
  public Double top() {
    return l.get(l.size()-1);
  }
  public Double pop() {
    return l.remove(l.size()-1);
  }
  public int size() {
    return l.size();
  }
}
```

Stack legyen általános!

sablonparaméter, formális típus

```
// írassunk ki visszafele
Stack<Double> s =
        new Stack<Double>();

s.push(1.0);
s.push(2.71);
s.push(3.14);

while (s.size()>0) {
  Double d = s.pop();
  System.out.println(d);
}
```

aktuális típus

```
public class Stack<T> {
  ArrayList<T> l;
  public Stack() {
    l = new ArrayList<T>();
  }
  public void push(T d) {
    l.add(d);
  }
  public T top() {
    return l.get(l.size()-1);
  }
  public T pop() {
    return l.remove(l.size()-1);
  }
  public int size() {
    return l.size();
  }
}
```

Ami tilos a sablonnál

- Primitív típust nem lehet

```
Stack<int> st = new Stack<int>(); // CT error
```

- Sablonparaméter nem példányosítható

```
T t = new T(); // CT error
```

- Sablonparaméter nem lehet statikus változó

```
static public T test; // CT error
```

```
Stack<String>.test = "hello";
Stack<Integer>.test = 13;     // CT error
```

Ami tilos a sablonnál

- Nem lehet a sablonparaméteren *instanceOf*

```
void foo(Stack<T> s) {
  if (s instanceof Stack<String>) ... // CT error
```

- Sablonból nem lehet tömböt csinálni

```
Stack<String>[] l1, l2, l3;
l1 = new Stack<String>[10];              // CT error
l2 = (Stack<String>[])new Object[10]; // CT error
l3 = (Stack<String>[])new Stack[10];  // OK
```

  - Kasztolást inkább ne csináljunk

Ami tilos a sablonnál

- Metódus polimorfizmus sablonparaméterrel
  - mi van, ha azonos típus lesz? (pl. *S* = *T* = *String*)

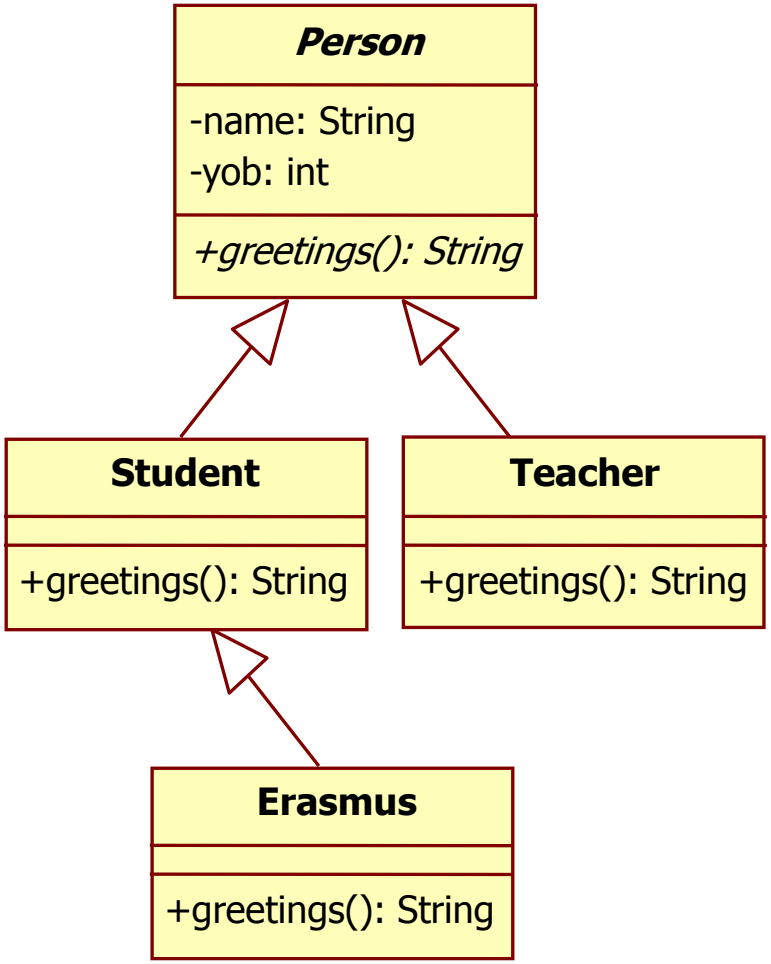
```
class Generic<S,T> {
   public void foo(S s) {...}
   public void foo(T t) {...} // CT error
```

- Kivételkezelésnél gond lehet
  - Jelzett kivétel nem lehet sablonparaméter
  - Catch-ben nem lehet elkapni sablonparamétert
  - El lehet dobni

Öröklésnél gondok vannak…

```
class Stack<T> {
  //...
  public void pushFrom(Stack<T> s) {
    for (T t : s.l) { l.add(t); }
  }
  public void pushTo(Stack<T> s) {
    for (T t : l) { s.l.add(t); }
  }
}
```

```
Stack<Person> sp = ...;
Stack<Student> ss = ...;
Stack<Erasmus> se = ...;
ss.pushFrom(se); // CT error
ss.pushTo(sp);   // CT error
```



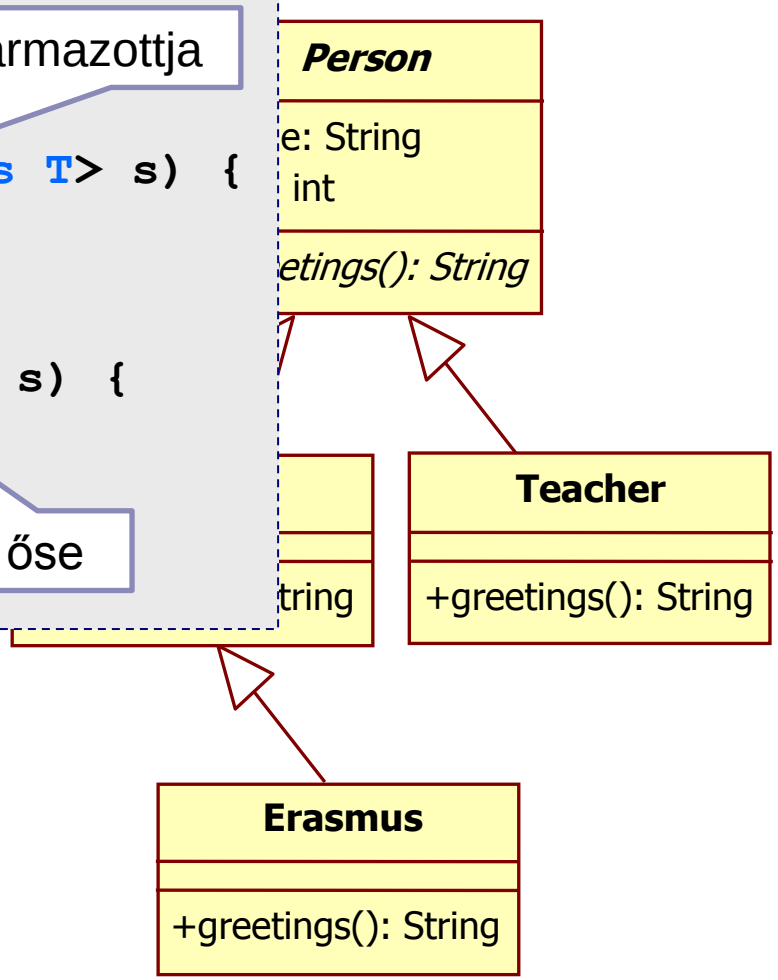
Öröklésnél *extends* és *super*

```
class Stack<T> {
  //...
  public void pushFrom(Stack<? extends T> s) {
    for (T t : s.l) { l.add(t); }
  }
  public void pushTo(Stack<? super T> s) {
    for (T t : l) { s.l.add(t); }
  }
}
```

bármi, ami *T* vagy leszármazottja

bármi, ami *T* vagy őse

```
Stack<Person> sp = ...;
Stack<Student> ss = ...;
Stack<Erasmus> se = ...;
ss.pushFrom(se); // OK
ss.pushTo(sp);   // OK
```



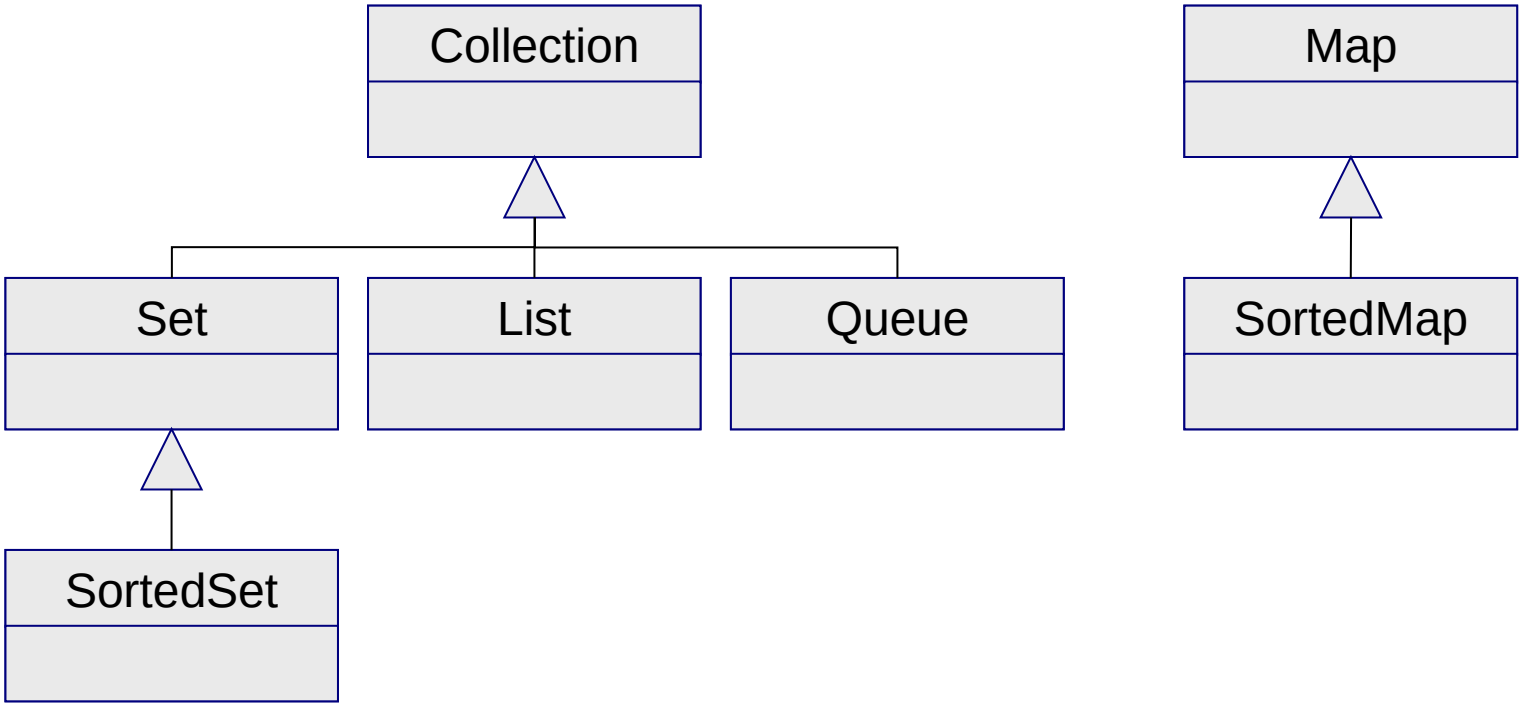
Kollekció keretrendszer (folytatás)

Kollekciók közös jellemzői (ism.)

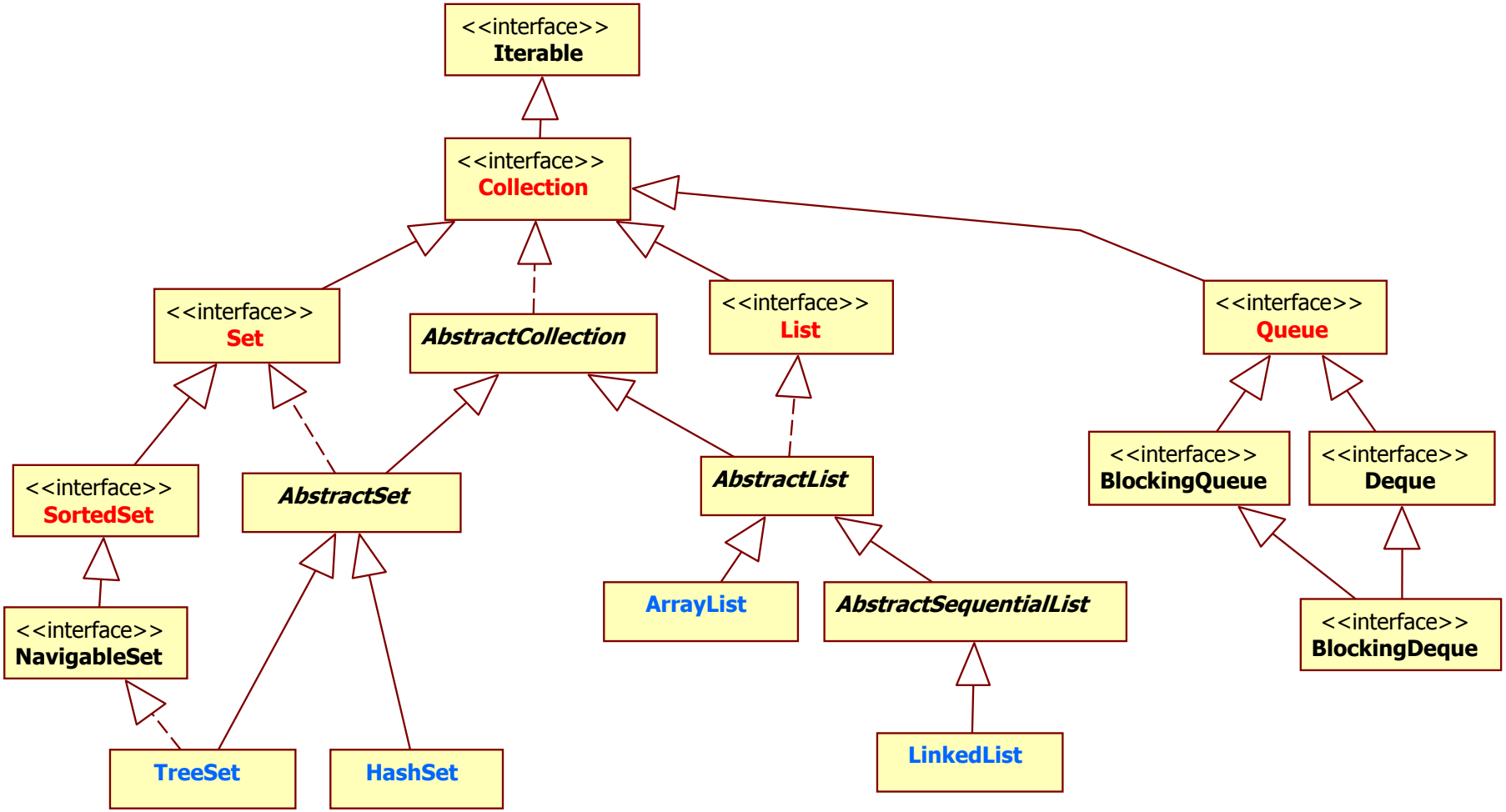
- Alapfunckiók
  - beszúrás, keresés, felülírás, törlés
    - CRUD: create, read, update, delete
  - iterálás
  - az elemek referenciáját tároljuk, kezeljük!
- Különböző megvalósítások
  - rendezett
  - halmaz vagy zsák
  - deep/shallow copy
  - különféle optimalizálások (pl. beszúrás vagy keresés)

Kollekciók (alapok, ism.)

- Interfészek



Kollekciók (bővebben, ism.)



Interface `List` (ism.)

- Sorrendben levő elemek (indexelt)
- Ugyanaz az elem többször is előfordulhat
- Elemek pozíciója ismert
  - indexeléssel elérhetők
- Kereshető (objektum -> index)
- Saját iterátor (`ListIterator`)
  - Iterator-t bővíti extra funkciókkal
- Tipikus megvalósítás: `ArrayList`

Interface `Set` (ism.)

- Set: halmaz, minden elem csak egyszer
- Sorrend nem definiált
- Iterator tetszőleges sorrendben jár be
- Nincs új metódusa
  - csak a Collection-ben definiáltak
- Tipikus megvalósítás: **`HashSet`**
  - jó hash függvény kell a hatékony működéshez

Interface `SortedSet`

- Halmaz rendezhető elemek számára
- Tartalom alapján rendez
  - természetes vagy **`Comparator`**-os
    - Természetes: **`Comparable.compareTo(Object o)`**
    - Comparator: **`int compare(Object o1, Object o2)`**
- **`Iterator`** rendezett sorban iterál
- Tipikus megvalósítás: **`TreeSet`**

Interface **`Map`**

- Kulcs-érték párok tárolása
- Kulcs alapján kereshető és módosítható
- Háromfajta nézetté alakítható
  - kulcsok
  - értékek
  - kulcs-érték párok
- Tipikus megvalósítás: **`HashMap`**

Interface **`Map`** 2

- Definíció: **`Map<K,V>`**
  - **`K`** kulcs típus
  - **`V`** érték típus
- Collection-ből ismert metódusok:
  - **`void clear()`**
  - **`boolean equals()`**
  - **`boolean isEmpty()`**
  - **`int size()`**

Interface `Map` 3

- Saját metódusok
  - `V put(K key, V value)`
    - kulcs-érték pár hozzáadása
  - `void putAll(Map<? ext K, ? ext V> m)`
    - sok kulcs-érték pár hozzáadása
  - `V get(K key)`
    - kulcs alapján érték keresése
  - `V remove(K key)`
    - kulcs alapján kulcs és érték törlése

Interface `Map` 4

- `boolean containsKey(Object key)`
- `boolean containsValue(Object value)`
  - igaz, ha kulcs ill. érték benne van
- `Set<K> keySet()`
  - kulcsok halmaza
- `Collection<V> values()`
  - értékek halmaza
- `Set<Map.Entry<K,V>> entrySet()`
  - kulcs-érték párok halmaza

Interface `SortedMap`

- Kulcsok alapján rendezve tárol
  - természetes vagy **`Comparator`**-os
  - hasonlóan a SortedSet-hez
- A nézetei is rendezve
  - **`keys(), values(), entrySet()`**
  - iteratorok rendezve iterálnak
- Tipikus megvalósítás: **`TreeMap`**

For-each ciklus

- Tömbökön is működik
- *Iterable* interfészű osztályokon használható
  - *public Iterator iterator()* metódus kell

```
Collection<Integer> c = getNumbers();
...
for (Integer i : c) {
        System.out.println(i);
}
```

```
static public void main(String[] args) {
  for (String s : args) { System.out.println(s); }
}
```

Kollekció példa

- Számoljunk betűgyakoriságot!

```
SortedMap<Character,Integer> map = new TreeMap<>();
Scanner sc = new Scanner(System.in);
while (sc.hasNextLine()) {
        String line = sc.nextLine();
        for (int i = 0; i < line.length(); i++) {
                char c = line.charAt(i);
                int n = 0;
                if (map.containsKey(c)) n = map.get(c);
                n++;
                map.put(c, n);
        }
}
for (char c : map.keySet()) {
        System.out.println("Char: " +c+"= \t"+map.get(c));
}
```

kulcsokon iterálunk

For-each-ciklus vagy Iterator?

- For-each-ciklus
  + olvashatóbb kód
  - kollekció nem változik
- Iterator
  + elem törlése támogatva
  - kód gazdagabb, olvashatóság rosszabb(?)

Kollekciók csomagolva

- Kollekció átadása gond lehet
  - a hívott metódus módosíthatja a tartalmat
- Megoldás: **`java.util.Collections`**
  - **`Collection unmodifiableCollection(Collection c)`**
  - **`List unmodifiableList(List c)`**
  - **`Map unmodifiableMap(Map c)`**
  - **`Set unmodifiableSet(Set c)`**
  - generikus típusokat is támogatja

Módosíthatatlan lista

```
// kapunk egy listát
List<Student> l = database.getStudents();

// átadjuk a listát, mi lesz a tartalmával?
doSomethingWithList(l);
```

```
// kapunk egy listát
List<Student> l = database.getStudents();

// becsomagoljuk: másolat nem keletkezik
List<Student> l2 = Collections.unmodifiableList(l);

// átadjuk a csomagot, ami továbbra is lista!
// módosító metódusok kivételt dobnak
// (UnsupportedOperationException)
checkForSomething(l2);
```

Input-output

Alap IO osztályok

	Input	Output
Char	*Reader* BufferedReader CharArrayReader FilterReader FileReader ...	*Writer* BufferedWriter CharArrayWriter FilterWriter FileWriter PrintWriter ...
Byte	*InputStream* ByteArrayInputStream FileInputStream FilterInputStream PipedInputStream ...	*OutputStream* ByteArrayOutputStream FileOutputStream FilterOutputStream PipedOutputStream ...

Reader metódusai

- int read()

  int read(char[] buf, *int off, int len*)
  - *len* karaktert olvas *buf*-ba *off* indextől
- mark(int limit), reset(), boolean markSupported()
  - jelölés és visszaléptetés
- skip(long n)
  - átlép *n* karaktert
- close()
  - lezárja az olvasást
- boolean ready()
  - igaz, ha a Reader-ből lehet olvasni

Writer metódusai

- write(int c)

  write(char[] buf, *int off, int len*)

  write(String s, *int off, int len*)
  - kiír *len* karaktert (lehet, hogy pufferel)
- flush()
  - a pufferelt tartalmat továbbítja
- close()
  - lezárja az írást

Speciális *Reader* osztályok

- BufferedReader
  - puffereli az olvasott karaktereket, sort is tud
- CharArrayReader / StringReader
  - char-tömbből vagy Stringből olvas
- FileReader
  - fájlból olvassa a karaktereket
- …

Reader példa

- Sorok olvasása fájlból *Scanner* nélkül
  - szabványos kimenetre írunk

```
FileReader fr = new FileReader(”hello.txt”);
BufferedReader br = new BufferedReader(fr);
while (true) {
      String line = br.readLine();
      if (line == null) break;
      System.out.println(line);
}
br.close();
```

Speciális *Writer* osztályok

- CharArrayWriter / StringWriter
  - char-tömbbe vagy String-be ír
  - van toString, toArray, size, stb. metódusa a további feldolgozáshoz
- FileWriter
  - fájlba ír
- PrintWriter
  - formázott kiíratás, több típust is támogat: *print*, *println*, *printf*, stb.
- …

Writer példa

- Írjuk ki az első 10 négyzetszámot!
  - formátum: **`X*X = Y`**

```
FileWriter fw = new FileWriter("squares.txt");
PrintWriter pw = new PrintWriter(fw);
//PrintWriter pw =
// new PrintWriter("squares.txt", "ISO-8859-1");
for (int i = 1; i <= 10; i++) {
      pw.println(i+"*"+i+" = "+(i*i));
      //pw.printf("%d*%d = %d\n", i, i, i*i);
}
pw.close()
```

InputStream metódusai

- int read(), int read(byte[] buf*, int off, int len*)
  - beolvas *len* byte-ot
- mark(int limit), reset(), boolean markSupported()
  - jelölés és visszalépés
- skip(long n)
  - kihagy *n* byte-ot
- close()
  - lezárja a stream-et
- boolean ready()
  - van-e olvasható byte?
- int available()
  - hány byte olvasható blokkolás nélkül?

OutputStream metódusai

- write(int c)

  write(byte[] buf, *int off, int len*)

  kiír *len* byte-ot
- flush()
  - puffert kimenetre üríti
- close()
  - lezárja a stream-et

Speciális InputStream-ek

- ByteArrayInputStream
  - byte-ot olvas tömbből
- FileInputStream
  - byte-ot olvas fájlból

Special OutputStreams

- ByteArrayOutputStream
  - a byte-okat egy tömbbe írja
- FileOutputStream
  - a byte-okat egy fájlba írja

Szabványos IO (ism)

- java.lang.System ismétlés
  - *InputStream in*
    - szabványos bemenet
    - byte alapú
  - *PrintStream out, err*
    - szabványos kimenet és hibakimenet
    - byte és char alapú

Szabványos IO példa

- Olvassunk stdin-t, írjuk stdout-ra
  - *java.util.Scanner* nélkül

```
InputStreamReader isr =
      new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
while (true) {
      String line = br.readLine();
      if (line == null) break;
      System.out.println(line);
}
br.close();
```

java.io.File

- Fájlok és könyvtárak (mappák) kezelése
  - triviális konstruktorok
    - String vagy File objektumokból
    - elérési utat adnak meg a paraméterek
  - OS függő információk
    - path szeparátor, mappa szeparátor
    - ne használjunk beégettett értéket ("/", "\", ";", ":")
  - file műveletek
    - törlés, temp. fájl létrehozása, hozzáférési jogok módosítása
  - file-ok adatai
    - létezik, név, szülő, hozzáférési jog, típus, méret, tartalom, …

File példa

- Aktuális mappa tartalmát írassuk ki rekurzívan!

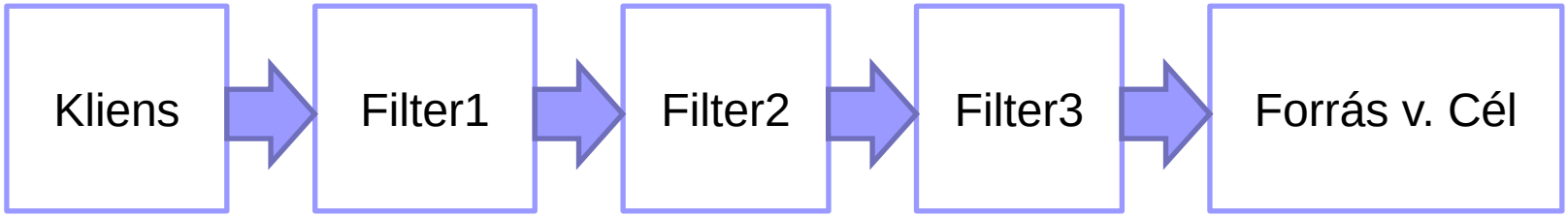
```
void lsdir(File f, String tab) {

   File[] list = f.listFiles();

   for (int i = 0; i < list.length; i++) {
      System.out.println(tab+list[i].getName());

      if (list[i].isDirectory()) {
         lsdir(list[i], tab+" ");
      }

   }
}
```

Java IO láncok

Kiíratás láncolt módon

- Alapötlet
  - azonos interfészű osztályok
  - különböző funkcionalitással
  - egymás metódusait hívják (delegáció)
    - láncolt listát képeznek
- Néha van extra funkcionalitás is
  - pl. *BufferedReader*
    - readLine()
- Általában konstruktorban kötjük össze
  - `new BufferedReader(new InputStreamReader(System.in))`

Láncolt működés



- Metódushívás balról jobbra
- Adatáramlás a típustól függ
  - reader / inputstream: balra
  - writer / outputstream: jobbra
- Lánc elemei nem tudják, hogy…
  - ki használja őket (ki a hívó)
  - akit használnak, mit csinál

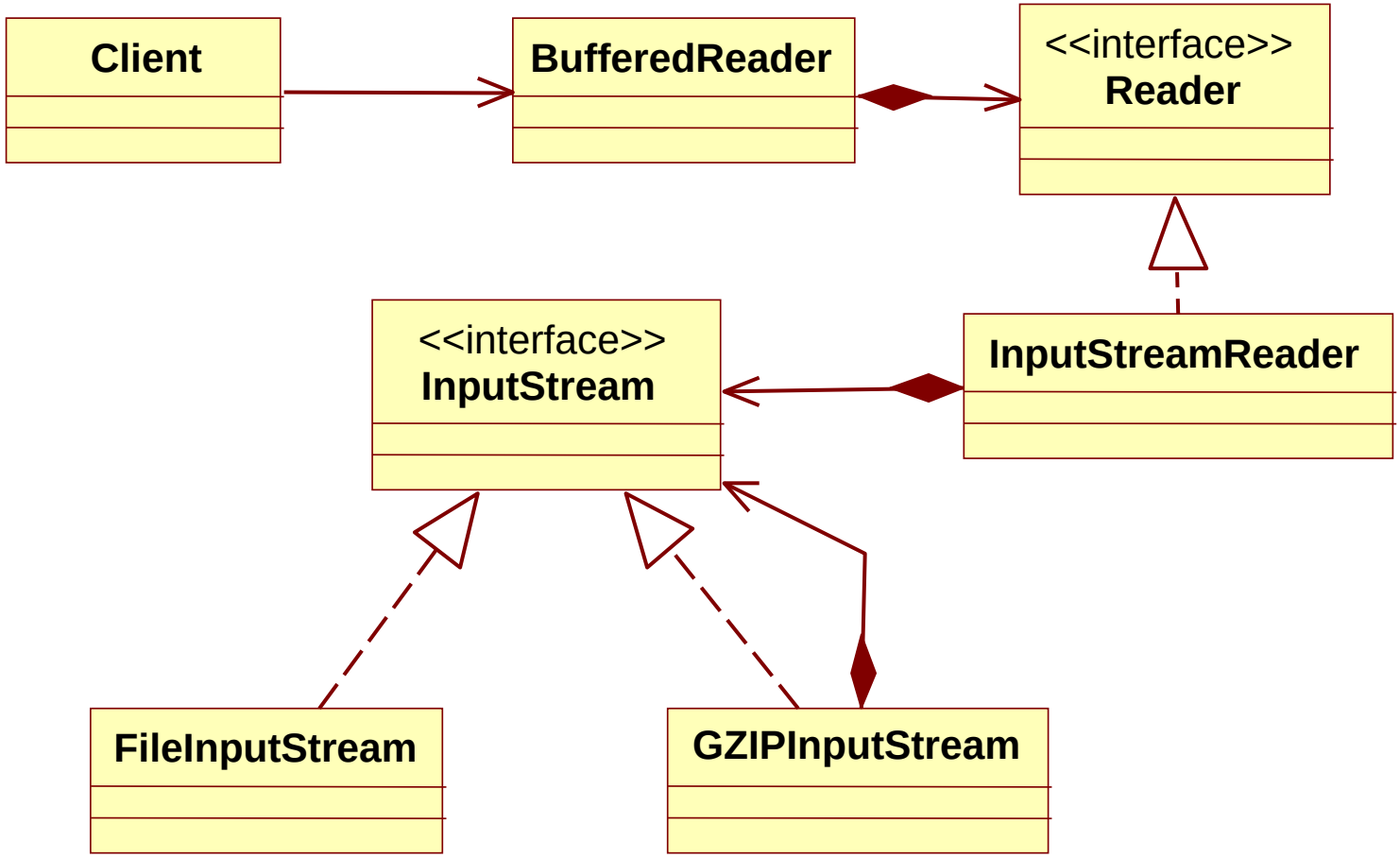
Pl: Char és Byte konverzió

- Olvasás
  - Forrás: *InputStream*
  - Kliens *Reader*-t vár
  - Megoldás: *InputStreamReader*
    - *Reader* interface, de *InputStream-ből olvas*
- Kiíratás
  - OutputStreamWriter
    - *Writer* interface, *OutputStream* cél

Pl: Kitömörítés

- GZIPInputStream
  - megvalósítja a GZIP kitömörítést
  - feladat: *írassuk ki a sorokat egy GZIP tömörített fájlból!*
- Megoldás
  - fájl tartalma: tömörített byte-ok
  - gzip kitömörítés: olvassa a byte-okat, de már kitömörítve adja őket vissza
  - sorokat kell kiírni:
    - kell byte-char konverzió
    - egy-egy char-ból sorokat kell építeni

Pl: Kitömörítés



Pl: Kitömörítés

```
// olvassuk a byte-okat
InputStream fis = new FileInputStream("test.gz");
// kitömörítjük
InputStream gis = new GZIPInputStream(fis);
// karakterré konvertáljuk
Reader isr = new InputStreamReader(gis);
// sorokká fűzzük
BufferedReader br = new BufferedReader(isr);

while (true) {
      String line = br.readLine();
      if (line != null) System.out.println(line);
      else break;
}
br.close()
```

Pl: Betömörítés

- GZIPOutputStream
  - feladat: *olvassunk sorokat és tömörítve írjuk ki fájlba*

```
BufferedReader br = ...;
PrintWriter pw = new PrintWriter(
   new OutputStreamWriter(
      new GZIPOutputStream(
         new FileOutputStream("test.gz"))));
while (true) {
      String line = br.readLine();
      if (line != null) pw.println(line);
      else break;
}
br.close(); pw.close();
```

Köszönöm a figyelmet