A programozás alapjai 3.

Java input-output

Goldschmidt Balázs
balage@iit.bme.hu

1

Java System osztály

2

System osztály

- `in, out, err`
  - statikus változók
  - szabványos be-, ki- és hibakimenetre referálnak
  - setter metódusokkal állíthatók
- `gc()`
  - garbage collector elindítása
- `exit(int)`
  - leállítja a virtuális gépet
- `getProperty/getProperties, setProperty...`
  - rendszerszintű beállítások elérése és módosítása
  - később visszatérünk rá

Java IO alapok

Java IO alapfogalmak

- Folyam (stream) alapú kommunikáció a külvilággal
  - UNIX örökség
- Kétfajta elemi információ
  - char → unicode, konverzió (karakterkészlet, sorvége)
  - byte → oktetek, nincs konverzió
- Szűrők használata
  - elemi funkcionalitás (tömörítés, konverzió stb.)
  - elemi funkcionalitásokból összetett megoldás építhető
- package java.io

https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/io/package-summary.html

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

- read()
  read(char[] buf*, int off, int len*)
  - beolvas 1 illetve *len* db. karaktert *buf*-ba *off* indextől
- mark(int limit), reset(), markSupported()
  - könyvjelző szolgáltatás
- skip(long n)
  - átlép *n* karaktert
- close()
  - bezárja a reader-t
- ready()
  - van-e olvasható karakter

Writer metódusai

- write(int c)
  write(char[] buf, *int off, int len*)
  write(String buf, *int off, int len*)
  - kiír 1 illetve *len* karaktert *buf*-ból *off* indextől
- flush()
  - a kimenetre küldi az átmeneti tárolóból a karaktereket
- close()
  - bezárja a writer-t

Speciális *reader*-ek

- BufferedReader
  - pufferelt beolvasás egész sorok beolvasására
- CharArrayReader / StringReader
  - char tömbből vagy string-ből olvas
- FileReader
  - fájlból olvas
- FilterReader
  - absztrakt osztály további reader-ek definiálásához

Reader példa

- Olvassunk sorokat
  - írjuk őket a szabványos kimenetre

```
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*-ek

- BufferedWriter
  - pufferelt kiíratás (hatékonyabb), *newLine()* is van
- CharArrayWriter / StringWriter
  - karakterek char tömbbe vagy string-be kerülnek
  - *toString*, *toArray*, *size* stb. elérhető
- FileWriter
  - karakterek fájlba írásához
- PrintWriter
  - formázott adatok kiírásához: *print*, *println*, *printf* stb.
- FilterWriter
  - absztrakt osztály további writer-ek definiálásához

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

- read(), read(byte[] buf, *int off, int len*)
  - beolvas 1 illetve *len* bájtot *buf*-ból *off* indextől
- mark(int limit), reset(), markSupported()
  - könyvjelző szolgáltatás
- skip(long n)
  - kihagy *n* bájtot
- close()
  - bezárja a stream-et
- ready()
  - van olvasható bájt
- available()
  - megadja, hogy hány olvasható bájt van

OutputStream methods

- write(int c)

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

  kiír 1 illetve *len* db. bájtot *buf*-ból *off* indextől
- flush()
  - a pufferelt bájtokat továbbítja a kimenetre
- close()
  - bezárja a stream-et

Speciális InputStream-ek

- ByteArrayInputStream
  - bájtok olvasása egy meglevő bájttömbből
- FileInputStream
  - bájtok olvasása fájlból
- FilterInputStream
  - absztrakt osztály további inputstream-ek definiálásához

Speciális OutputStream-ek

- ByteArrayOutputStream
  - bájtok írása egy bájttömbe
- FileOutputStream
  - bájtok írása egy fájlba
- FilterOutputStream
  - absztrakt osztály további outputstream-ek definiálásához
- PrintStream
  - különféle adatok kényelmes kiírásához

Szabványos IO

- java.lang.System ismét
  - *InputStream in*
    - szabványos bemenet
    - bájtokat ad ☹
  - *PrintStream out, err*
    - szabványos kimenet és hibakimenet
    - bájt és karakter egyszerre támogatva

Szabványos IO példa

- Olvassunk a bemenetről, írjunk a kimenetre!
  - *java.util.Scanner* jobb megoldás, később előkerül ☺

```
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

- Fájlrendszer eléréséhez (fájlok, mappák)
  - triviális konstruktorok
    - String vagy File paraméterezéssel
    - elérési úttal, fájlnévvel, kombináltan is
  - operációs rendszertől függő adatok
    - elérési útvonal elválasztója, mappák közötti szeparátor stb.
    - ne használjuk ezeket: "/", "\", ";", ":"
  - fájlműveletek
    - törlés, ideiglenes fájl létrehozása, hozzáférési jogok állítása
  - fájl-adatok
    - létezik-e, nevei, mappája, jogai, típusa, mérete, tartalma

File példa

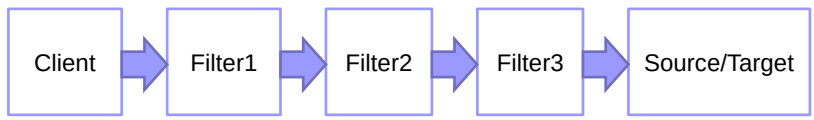
- Listázzuk ki rekurzívan a megadott mappát!

```
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+" ");
      }

   }
}
```

Szűrő minta (decorator)

- Az alapötlet
  - azonos interfész (metódusok)
  - egyedi működés
  - megvalósításban a partner meghívása (delegáció)
    - ezzel egy objektum-láncot kapunk
- Néha extra metódusok is megjelennek
  - pl. *BufferedReader*
    - readLine()
- Általában konstruktorban kapcsoljuk össze őket
  - `new BufferedReader(new InputStreamReader(System.in))`

Szűrő minta használata



- Hasonló a UNIX csővezetékhez (pipeline)
  - `ls -l | grep "^d" | sort -k 9 | tail -n 3`
- Az adatfolyam iránya read/write alapon
  - reader / inputstream: balra
  - writer / outputstream: jobbra
- A szűrők nem tudják, hogy …
  - ki használja őket
  - mit csinál, akit ők hívnak

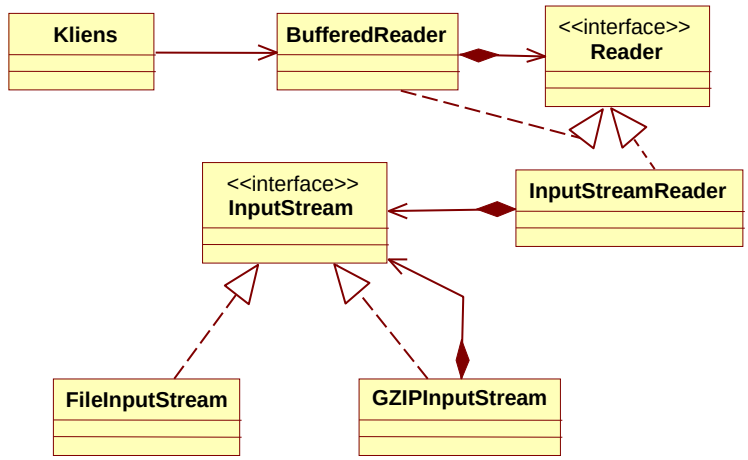
Példa: char és byte konvertálása

- Olvasás
  - Forrás: *InputStream*
  - Kliens *Reader*-t szeretne
  - Megoldás: *InputStreamReader*
    - *Reader* interfést, de *InputStream* forrás
- Kiírás
  - OutputStreamWriter
    - *Writer* interfész, *OutputStream* cél

Példa: tömörítés 1

- GZIPInputStream
  - GZIP kitömörítést biztosít
  - feladat: *írjuk ki egy gzip tömörített szövegfájl tartalmát*
    - *unix-ban:* `zcat f.txt.gz`
  - fájl tartalma: bájtok
  - gzip kitömörítés: olyan szűrő, ami bájtokat olvas, bájtokat ad, és közben kitömöríti a tartalmat
  - sorokra van szükségünk:
    - byte-char konvertálás
    - karakterekből sorok fűzése

Példa: tömörítés 1



25

Példa: tömörítés 1

```
// bájtok olvasása fájlból
InputStream fis = new FileInputStream("test.gz");
// kitömörítés
InputStream gis = new GZIPInputStream(fis);
// bájtok konvertálása karakterré
Reader isr = new InputStreamReader(gis);
// karakterek összefűzése sorokká
BufferedReader br = new BufferedReader(isr);

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


26

Példa: tömörítés 2

- GZIPOutputStream
  - feladat: *olvassunk sorokat és írjuk tömörített 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();
```


Saját szűrő készítése

- FilterInputStream, FilterOutputStream
- FilterReader, FilterWriter
  - ugyanaz az interfész és ősosztály
  - alapból mindent delegálnak a metódusok

```
public int write(byte[] buf, int off, int len)
throws IOException {
      return out.write(buf, off, len);
}
```

Saját szűrő készítése

- Származtassuk le a FilterXXX osztályt
  - metódusokat úgy készítjük, ahogy akarjuk
  - használjuk az *in* vagy *out* attribútumot delegáláshoz
  - ne feledjük a konstruktort!

```
public class MyInFilter
extends java.io.FilterInputStream {
      public MyInFilter(InputStream arg0) {
            super(arg0);
      }
      ...
}
```


Szűrő példa: CryptoIS/1

```
public class CryptoIS extends FilterInputStream {
  int key;

  public CryptoIS(InputStream arg0, int k) {
    super(arg0);
    key = k;
  }
  private int convert(int c) {
    return c^key; // titkosítás xor-ral (példa)
  }
  public boolean markSupported() {
    return false;
  }
//...
```

Szűrő példa: CryptoIS/2

```
//...
  public int read() throws IOException {
      int a = in.read();
      return (a<0) ? a : convert(a);
  }
  public int read(byte[] cbuf, int off, int len)
  throws IOException {
    int ret = in.read(cbuf, off, len);
    for (int i = 0; i < ret; i++) {
      cbuf[i+off] = (byte)convert(cbuf[i+off]);
    }
    return ret;
  }
}
```


Szűrő példa: CryptoOS/1

```
public class CryptoOS extends FilterOutputStream {
  int key;
  public CryptoOS(OutputStream arg0, int k) {
    super(arg0);
    key = k;
  }

  private int convert(int c) {
    return c^key;
  }
  //...
```

Szűrő példa: CryptoOS/2

```
  //...
  public void write(int b) throws IOException {
            out.write(convert(b));
  }
  public void write(byte[] cbuf, int off, int len)
  throws IOException {
    for (int i = 0; i < len; i++) {
      cbuf[i+off] = (byte)convert(cbuf[i+off]);
    }
    out.write(cbuf, off, len);
  }
}
```


Szűrő példa: használat (fájlba)

```
// code snippet
try {
  OutputStream os =
    new FileOutputStream("test.txt");

  os = new CryptoOS(os, 13);

  Writer w =
    new OutputStreamWriter(os);
  PrintWriter pw = new PrintWriter(w);

  BufferedReader br = new BufferedReader(
    new InputStreamReader(System.in));
  //...
```

Szűrő példa: használat (fájlba)

```
  //...

  String line;
  while (true) {
    line = br.readLine();
    if (line == null) break;
    pw.println(line);
  }
  br.close(); // csak a legutolsó szűrőn hívjuk
  pw.close(); // csak a legutolsó szűrőn hívjuk
} catch (Exception e) {
  e.printStackTrace();
}
```


Szűrő példa: használat (fájlból)

```
// kódrészlet
try {

  InputStream is = new
    FileInputStream("test.txt");

  is = new CryptoIS(is, 13);

  Reader r = new InputStreamReader(is);
  BufferedReader br = new BufferedReader(r);

  //...
```

Szűrő példa: használat (fájlból)

```
  //...
  String line;
  while (true) {
    line = br.readLine();
    if (line == null) break;
    System.out.println(line);
  }
  br.close(); // only called on outermost filter

} catch (Exception e) {
  e.printStackTrace();
}
```


Szűrő példa: tartalom

h	e	l	l	o		w	o	r	l	d	!	\r	\n

e	h	a	a	b	-	z	b	177	a	i	,	\0	\a
101	104	97	97	98	4	122	98	127	97	105	44	0	7
65	68	61	61	62	2d	7a	62	7f	61	69	2c	00	07

Szerializálás

Bevezetés

- Objektumok a memóriában vannak
  - hogyan küldjük át őket a hálózaton?
  - hogyan tároljuk őket fájlokban?
- Strukturált adatreprezentációk
  - Sorosítás
  - XML
  - JSON

Sorosítás alapok

- Feladat
  - mentsük el az objektumokat, majd töltsük vissza őket
  - mit kell tárolni?
    - objektumok attribútumai
    - asszociációk
    - statikus mezők?
- Egyszerű megoldás: *sorosítás (serialization)*
  - Java beépítetten tartalmazza
  - objektumok adatait és asszociációit bájtfolyamba tudja menteni és onnan visszatölteni

Sorosítás fogalmak

- Sorosítás (serialization)
  - objektumok konvertálása bináris formába (bájtfolyam)
    - további angol elnevezések: marshalling, deflating
  - bináris adat tárolható, hálózaton átküldhető, stb.
- Visszasorosítás (deserialization)
  - bináris adat (bájtfolyam) konvertálása objektumokká
    - további angol elnevezések: unmarshalling, inflating
  - bináris adat olvasható fájlból, érkezhet hálózaton, stb.

Sorosítás példa: mentés

```
import java.io.Serializable;
public class SerializableClass implements Serializable
{ ... }
```

```
//import java.io.*;
...
SerializableClass ser = ...;
try {
      FileOutputStream f =
                  new FileOutputStream(”filename”);
      ObjectOutputStream out =
                  new ObjectOutputStream(f);
      out.writeObject(ser);
      out.close();
}
catch(IOException ex) { ... }
```


Sorosítás példa: visszaolvasás

```
//import java.io.*;
...
SerializableClass ser2;
try {
      FileInputStream f =
                  new FileInputStream("filename");
      ObjectInputStream in =
                  new ObjectInputStream(f);
      ser2 = (SerializableClass)in.readObject();

      in.close();

} catch(IOException ex) {
      ...
} catch(ClassNotFoundException ex) {
      ...
}
```

Sorosítás szabályai

- Csak olyan osztályok sorosíthatók, amelyek implementálják a *Serializable* interfészt
  - az is elég, ha az ősosztály implementál
  - tömbök, String, Integer, Double, stb. sorosíthatók
- *Serializable* interfész
  - nincsenek metódusai
  - csak egy jelzés a fordítónak
- *Nem sorosíthatók:*
  - *Object, Socket, Input/OutputStream, System, stb.*

Sorosítás szabályai

- Mi lesz sorosítva?
  - primitív típusú attribútumok
  - sorosítható típusú attribútumok
    - rekurzívan
- Mi nem lesz sorosítva?
  - statikus mezők
  - *transient* mezők

```
public class Serial implements Serializable {
       transient private String secret;
       private String other;
       ...
```

Sorosítás folyamata

- A sorosítás rekurzív
  - hogyan kerüljük el a köröket az objektumgráfban?
  - az objektum tartalma csak egyszer íródik ki
  - minden további alkalommal csak egy referencia

```
out.writeObject(a);
out.writeObject(b);
out.writeObject(a); // csak referencia!
```

- Örökölt tagok sorosítása
  - a legfelső sorosítható őstől indul

Sorosítás folyamata 2

- Vegyük a következő osztályokat!

```
class Student implements
    Serializable {
  String name;
  University uni;
  Address a;
  double average;
  // ctr, set, get
}
```

```
class Address implements
    Serializable {
  String country,
    city, street;
  // ctr, set, get
}
```

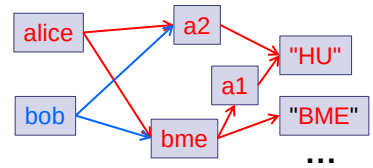
```
class University implements
    Serializable {
  String name;
  Address a;
  // ctr, set, get
}
```

Sorosítás folyamata 3

- Vegyük a következő objektumokat!

```
Address a1 = new Address("HU", "Bp", "Műegyetem rkp 3");
Address a2 = new Address("HU", "Bp", "Alkotás u. 10");

University bme = new University("BME", a1);

Student alice = new Student("Alice", bme,  a2);
Student bob   = new Student("Bob", bme, a2);
```

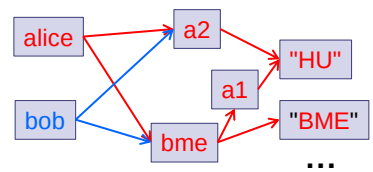
```
ObjectOutputStream oos = ...;
oos.writeObject(alice);
oos.writeObject(bob);
oos.close();
```



Sorosítás folyamata 4

- Olvassunk vissza!

```
ObjectInputStream ois = ...;
Student alice2 = (Student)ois.readObject();
Student bob2 = (Student)ois.readObject();
ois.close();

System.out.println(bob2.getUni().getName());
```



Verziózás

- Minden osztálynak egyedi verzióazonosítója van
  - lekérdezés: `serialver` *`ClassName`*
- Kompatibilitás biztosítása
  - ugyanannak a verzióazonosítónak (ID) a használata:

```
static final long serialVersionUID
 = 1027553947283749 5L;
```

- Kompatibilis változások
  - metódus/mező hozzáadása vagy láthatóság módosítása
  - static/transient → perzisztens

Köszönöm a figyelmet!