Adatfolyamok, állománykezelés, szerializáció

Simon Károly
simon.karoly@codespring.ro

Adatfolyamok

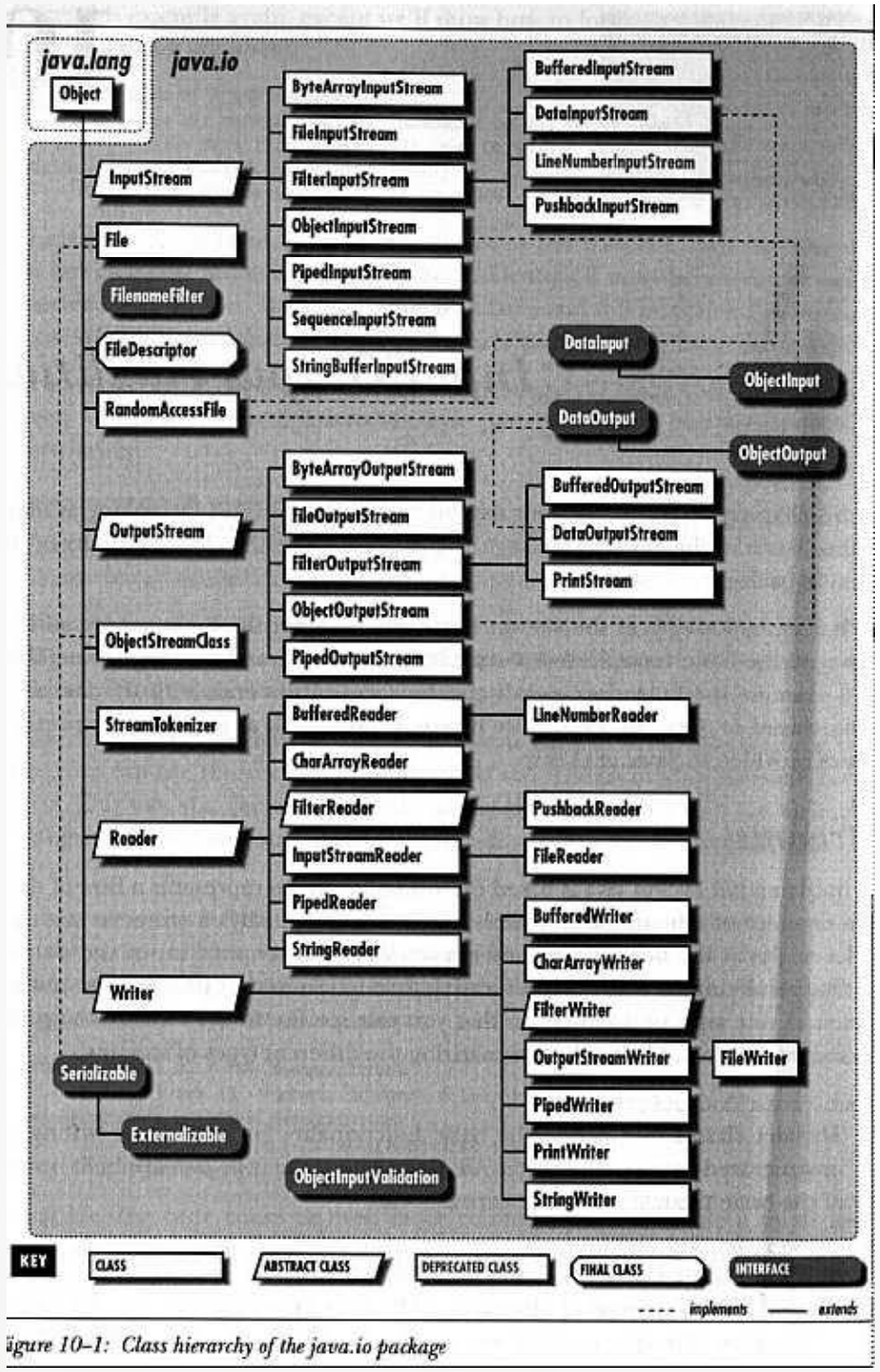

Figure 10–1: Class hierarchy of the java.io package

- Cél: számítástechnikai rendszereken belüli entitások közötti kommunikáció
- Adatfolyam (stream): két entitás közötti kommunikációs csatorna
- Forrás → cél: egyirányú átvitel
- → bemeneti és kimeneti adatfolyamok (input stream/output stream)
- Példa: billentyűzetnek (input), illetve monitornak (output) megfeleltetett stream
- Adatfolyamok összekapcsolása → pipe-line mechanizmus → folyamatok közötti kommunikáció
- Bájt és karakter típusú adatfolyamok (bináris/szöveges stream-ek)

java.io

- **InputStream/OutputStream**: bináris adatfolyamokkal kapcsolatos alapfunkcionalitásokat (írás/olvasás) biztosító alaposztályok, a további osztályok ezekből származnak
- **Reader/Writer**: szöveges adatfolyamokkal kapcsolatos alapfunkcionalitásokat (írás/olvasás) biztosító alaposztályok, a további osztályok ezekből származnak
- **InputStreamReader/InputStreamWriter**: bináris és szöveges adatfolyamok közötti konverzió
- **DataInputStream/DataOutputStream**: az InputStream és OutputStream –ekre alkalmazható szűrők, amelyek különböző adattípusok olvasására/írására adnak lehetőséget
- **ObjectInputStream/ObjectOutputStream**: szerializálható objektumok írása/olvasása
- **BufferedInputStream/BufferedOutputStream**: átmeneti (puffer-) zóna alkalmazása
- **PrintWriter**: karakterek írására alkalmazható speciális adatfolyam (ilyen a System.out)
- **PipedInputStream/PipedOutputStream/PipedReader/PipedWriter** – a pipeline mechanizmus megvalósításához
- **FileInputStream/FileOutputStream/FileReader/FileWriter**: állománykezelés

Konzol I/O

- ```
  InputStream    in
  PrintWriter    out
  PrintWriter    err
  ```
- Olvasás:

```
int read() throws IOException;
int read(byte b[]) throws IOException;
int read(byte b[], int offset, int length) throws IOException;
```

```
try {
    int val = System.in.read();
    ...
} catch(IOException e ) {
    // kivétel kezelés
}
```

```
byte b[1024];
try {
  int bytesread=System.in.read(b);
} catch(IOException e) {...}
```

- Adatfolyamon belüli oktettek száma:

```
try {
    int available = System.in.available();
    if( available > 0) {
        byte b[] = new byte[available];
        System.in.read(b);
    }
} catch( IOException e ){ ... }
```
```

Példa

- A standard bemenet standard kimenetre másolása:

```
import java.io.*;
public class StreamExample {
   public static void copy(InputStream sin, OutputStream sout)
                              throws IOException {
      int b;
      while((b = sin.read()) != -1) sout.write(b);
      sout.flush();
   }
   public static void main(String[] args) {
      try {
               copy(System.in, System.out);
      } catch(IOException e) {
               System.out.println("Masolasi hiba");
      }
   }
}
```

Primitív adatok írása/olvasása

- A DataInputStream és DataOutputStream metódusai:
  - ```
    short readShort();
    int readInt();
    long  readLong();
    float readFloat();
    double readDouble();
    ```
    ```

Állománykezelés

- A FileInputStream, FileOutputStream, FileReader, FileWriter osztályok
- **Példa:** egy szöveges állomány megjelenítése a konzolon:

```
import java.io.*;
public class Cat {
   public static void main( String args[]) {
      FileReader fr = null;
      int b = 0;
      if(args.length != 1) {
               System.out.println("Inditas: java Cat <fajlnev_text>");
               System.exit(1);
       }
       try {
               fr = new FileReader(args[0]);
               while((b = fr.read()) != -1)
                        System.out.print((char) b);
       } catch(FileNotFoundException e) {
               System.out.println("Nemletezo allomany");
               System.exit(2);
       } catch(IOException e) {
               System.out.println("Olvasasi hiba");
               System.exit(3);
      }
   }
}
```

Állománykezelés - példa

- **Példa:** szöveges állomány, egész sorok beolvasása:

```
try {
        fr = new FileReader(args[0]);
        BufferedReader br = new BufferedReader(fr);
        String line;
        while((line = br.readLine()) != null)
                System.out.println(line);
} catch(FileNotFoundException e) {
        System.out.println("Nemletezo allomany");
        System.exit(2);
} catch(IOException e) {
        System.out.println("Olvasasi hiba");
        System.exit(3);
}
```

A File osztály

- Fontosabb metódusok:

Metódus	***Visszafordított típus***	***Leírás***
canRead()	*boolean*	*Engedélyezett az olvasás?*
canWrite()	*boolean*	*Engedélyezett az írás?*
delete()	*boolean*	*Törlés*
exists()	*boolean*	*Létezik a fájl?*
getAbsolutePath()	*String*	*A teljes elérési út*
getCanonicalPath()	*String*	*A teljes elérési út*
getName()	*String*	*A fájl neve*
getParent()	*String*	*A könyvtár neve*
getPath()	*String*	*Az elérési út*
isDirectory()	*boolean*	*Könyvtár?*
isFile()	*boolean*	*Fájl?*
lastModified()	*long*	*Utolsó módosítás*
length()	*long*	*A fájl hossza*
list()	*String[]*	*A könyvtáron belüli fájlok listája*
mkdir()	*booolean*	*Könyvtár létrehozása*

File osztály - példa

- Fájl tartalmának megjelenítése,
  vagy a könyvtárban található fájlok listázása:

```
import java.io.*;
public class FileExample {
  public static void main(String args[]) {
      File file;
      if (args.length != 1) {
              System.out.println("Indítás: java Listazas <nev> ");
              return;
      }
      try {
              file = new File(args[0]);
      } catch(Exception e) {
              System.out.println("Nemlétező állomány ");
              return;
      }
      if (!file.exists() || !file.canRead()) {
              System.out.println("Olvasási hiba: "+ args[0]);
              return;
      }
```

File osztály - példa

```
if (file.isDirectory()) {
        String files[] = file.list();
        for(int i = 0; i < files.length; i++)
                System.out.println(files[i]);
}
else {
        try {
                FileReader fr = new FileReader(file);
                BufferedReader in = new BufferedReader(fr);
                String line;
                while((line = in.readLine()) != null)
                        System.out.println(line);
            } catch(FileNotFoundException e) {
                System.out.println("Nemlétező állomány");
            } catch(IOException e) {
                System.out.println("olvasási hiba");
            }
        }
    }
}
```

A RandomAccessFile osztály

- ```
  try {
      RandomaccessFile file = new RandomAccessFile("fajl","rw");
      ...
  } catch(IOException e) { }
  ```
- Példa: valós számokat tartalmazó állomány létrehozása, véletlenszerű feltöltése, tartalmának megjelenítése, majd néhány érték véletlenszerű megváltoztatása:

```
import java.io.*;
import java.util.*;

public class RAFExample {
   private RandomAccessFile f;
   private String  name;
   private long    dim;
   public RAFExample(String name, long dim) throws IOException {
      this.name = name;
      this.dim  = dim;
      f = new RandomAccessFile(name, "rw");
      Random r = new Random();
      for (int i = 0; i < dim; i++) {
                double d = r.nextDouble();
                f.writeDouble(d);
                System.out.print(d+"\t");
      }
      f.close();
      f = new RandomAccessFile(name, "rw");
   }
```
```

A RandomAccessFile osztály

```
public double getDouble(long poz) throws IOException {
        f.seek(poz * 8);
        return(f.readDouble());
}
public void putDouble(long poz, double d) throws IOException {
        f.seek(poz * 8);
        f.writeDouble(d);
}
public long getDim() {return dim;}
public static void main(String args[]) {
        try {
                RAFExample c = new RAFExample("doublefile",10);
                Random r = new Random();
                long n = c.getDim();
                for(int i = 0; i < 5; i++) {
                        long poz = r.nextLong();
                        if (poz < 0) poz = -poz;
                        poz = poz % n;
                        System.out.println("Poz: " + poz);
                        c.putDouble(poz, 1.0);
                }
                System.out.println("A megvaltoztatott allomany: ");
                for(long i = 0; i < n; i++) System.out.print(c.getDouble(i) +"\t");
        } catch(IOException e) {
                System.out.println("Hiba ");
        }
    }
```

Java Serialization API

- Standard eljárás az objektumok állapotának adatfolyamba történő kiírására (elmentésére egy bájtszekvenciába) és visszatöltésére
- Perzisztencia (archiválás későbbi felhasználásra), osztott OO rendszerek (távoli metódushívások) (marshalling – szerializáció, a codebase elmentésével)
- Lehetőségek: alapértelmezett protokoll, vagy annak módosítása, saját protokoll
- ObjectInputStream, ObjectOutputStream
- Kiírás

```
fouts = new FileOutputStream(filename);
out = new ObjectOutputStream(fouts);
out.writeObject(new MyObject());          //serialization
out.close();
```

- Beolvasás

```
fis = new FileInputStream(filename);
in = new ObjectInputStream(fis);
time = (MyObject)in.readObject();         //live object
                                          //exact replica
in.close();
```

- Természetesen try-cath blokkon belül (IOException, FileNotFoundException, NotSerializableException (írás), ClassNotFoundException (beolvasás)).

Serializable interface

- Az objektumoknak implementálniuk kell a Serializable interface-t (vagy az alaposztálytól örökölniük az implementációt).
- Az Object nem implementálja, tehát nem minden objektum szerializálható (de a legtöbb standard Java osztály – alaposztályok, listák, GUI komponensek – igen).
- Nem szerializálható objektumok pl.: thread, socket, stream stb. (de attól, hogy egy osztály tartalmaz pl. egy thread példányt, a többi része még lehet szerializálható → transient típusmódosító alkalmazása).
-
```
import java.io.Serializable;
import java.util.Date;
import java.util.Calendar;
public class PersistentTime implements Serializable {
     private Date time;
     public PersistentTime() {
             time = Calendar.getInstance().getTime();
     }
     public Date getTime(){
             return time;
     }
}
```

Transient

- Azokat az adattagokat, amelyeket nem lehet szerializálni, vagy nem szeretnénk, hogy részei legyenek az objektum perzisztens állapotának (nem akarjuk elmenteni) **transient** –nek nyilvánítjuk.
- ```
  import java.io.Serializable;
  public class PersistentAnimation implements Serializable, Runnable {
        transient private Thread animator;
        private int animationSpeed;
        public PersistentAnimation(int animationSpeed) {
                this.animationSpeed = animationSpeed;
                animator = new Thread(this);
                animator.start();
        }
        public void run(){
                while(true) {
                 // do animation here
                }
        }
  }
  ```
  ```

Protokoll testreszabása

- Példa: az animáció esetében, ha elmentünk, majd beolvasunk egy PersistentAnimation példányt, a konstruktor nem kerül meghívásra (nem történik példányosítás), az objektum nem fog megfelelő módon viselkedni (a szál nem jön létre, nem indíthatjuk el).
- Ha külön metódusba tennénk a problémás részt, a felhasználónak tudnia kellene erről a metódusról.
- Jobb megoldás a protokoll módosítása, a következő metódusok segítségével:

```
private void writeObject(ObjectOutputStream out)
                                              throws IOException;
private void readObject(ObjectInputStream in)
                        throws IOException, ClassNotFoundException;
```

- A metódusok első sorai:

```
out.defaultWriteObject();
in.defaultReadObject();
```

  Nem implementáljuk a szerializálás mechanizmust, csak kiterjesztjük azt (hozzáadunk).

Protokoll testreszabása

```
import java.io.Serializable;
public class PersistentAnimation implements Serializable, Runnable {
      transient private Thread animator;
      private int animationSpeed;
      public PersistentAnimation(int animationSpeed) {
            this.animationSpeed = animationSpeed;
            startAnimation();
      }
      public void run() {
           while(true) {
            // do animation here
           }
      }
      private void writeObject(ObjectOutputStream out) throws IOException {
            out.defaultWriteObject();
      }
      private void readObject(ObjectInputStream in)
                                    throws IOException, ClassNotFoundException {
            in.defaultReadObject();
            startAnimation();
      }
      private void startAnimation() {
            animator = new Thread(this);
            animator.start();
      }
}
```

Letiltás

- Amennyiben nem szeretnénk, hogy az osztályunk példányai szerializálhatóak legyenek (bár az alaposztály implementálja az interfészt, és mi nem „unimplementálhatjuk”):

```
private void writeObject(ObjectOutputStream out) throws IOException {
     throw new NotSerializableException("Not today!");
}

private void readObject(ObjectInputStream in) throws IOException {
     throw new NotSerializableException("Not today!");
}
```

Egyéni protokoll

- Az Externalizable interface megvalósítása

- `public void writeExternal(ObjectOutput out) throws IOException;`
- `public void readExternal(ObjectInput in)`
  `throws IOException, ClassNotFoundException;`

- Semmi nincs implementálva, a megvalósítás teljes egészében a mi feladatunk.
- A metódusok a writeObject és readObject metódushívások esetén automatikusan meghívásra kerülnek (hasonlóan az előző példákhoz).
- Példa: speciális fájlformátumoknak (pl. pdf) megfelelő Java objektumok szerializálása.

Version controll

- Lementjük egy objektum állapotát, módosítjuk az osztályt, majd vissza akarjuk tölteni az elmentett objektumot → InvalidClassException (minden szerializálható osztályhoz egy egyedi azonosító rendelődik hozzá, ha ez nem talál, kivételt kapunk).
- Ha például csak egy adattagot adtunk hozza → szeretnénk, hogy ez ne így történjék, az illető adattag legyen inicializálva az alapértelmezett értékkel és történhessen meg a betöltés.
- Megoldás: az azonosító a serialVersionUID mezőben van tárolva → ennek értékét beállíthatjuk manuálisan (a JDK biztosít nekünk ehhez egy serialver nevű eszközt, ami az azonosítót automatikusan generálja, alapértelmezetten az objektum hash kódja alapján).
- Kompatibilis változtatások: pl. attribútum/metódus törlése/hozzáadása.
- Inkompatibilis változtatások: pl. hierarchia megváltoztatása, interfész implementálásának eltávolítása (pl. Serializable).
- A különböző (típusú) változtatások listája megtalálható a Java Serialization Specification –ben.

Objektum caching

- Az ObjectOutputStream alapértelmezetten fent tart egy a beléje írt objektumra mutató referenciát → ha kiírjuk az objektumot, majd megváltoztatjuk és újra kiírjuk, az új állapot nem lesz lementve.
- Példa:

```
ObjectOutputStream out = new ObjectOutputStream(...);
MyObject obj = new MyObject(); // must be Serializable
obj.setState(100);
out.writeObject(obj); // saves object with state = 100
obj.setState(200);
out.writeObject(obj); // does not save new object state
```

- Megoldás(ok):
  - Mindig zárjuk a streamet írás után, vagy
  - használjuk az ObjectOutputStream.reset() metódust (vigyázat: a metódus az összes tárolt referenciát felszabadítja, törli a cache-t).

Feladat

- A SWING eszközkészlettel kapcsolatos feladatot (alakzatok kirajzolása SWING felületen) egészítsük ki mentési és betöltési lehetőséggel. A megfelelő menük segítségével a felhasználó kérheti egy adott konfiguráció (alakzat típusa, színe, mérete) elmentését, illetve betöltését. A cél- és forrásállományok kiválasztásában *JFileChooser* komponensek szolgáltatnak segítséget. A betöltés esetében az alakzat megjelenítésén kívül a komponensek állapotát is változtassuk.