Ingineria Sistemelor de Programare

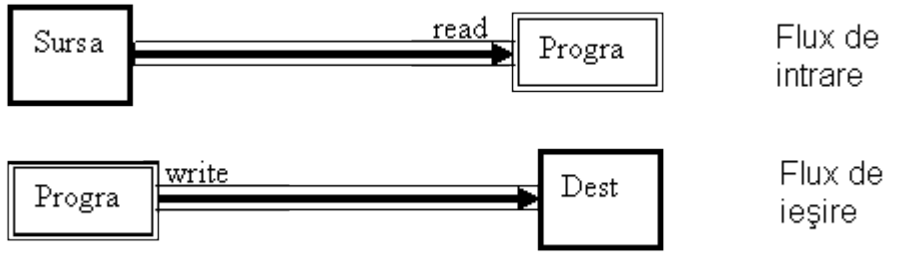
Fluxuri IO

mihai.hulea@aut.utcluj.ro

2017

Introducere

- Fluxuri IO (eng. IO Strems)= canale de comunicatie unidirectionale sau bidirectionale utilizate pentru citirea respective scrierea de informatii de catre un program
- Clasele Java pentru lucrul cu fluxuri se gasesc in pachetele:
  - *java.io*
  - *java.nio*



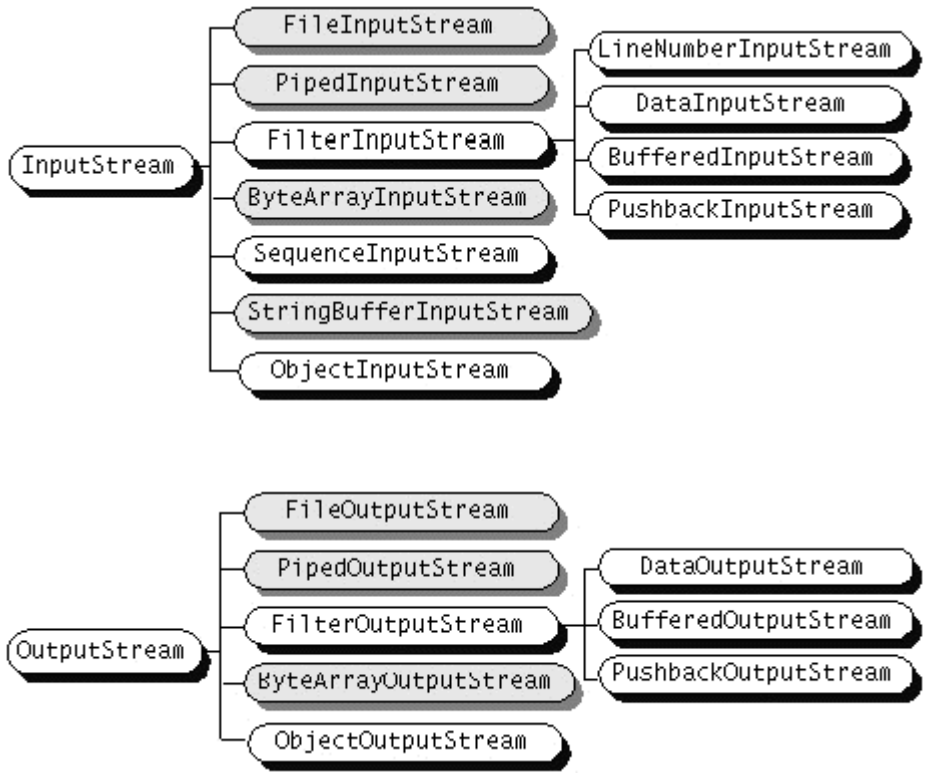
Introducere

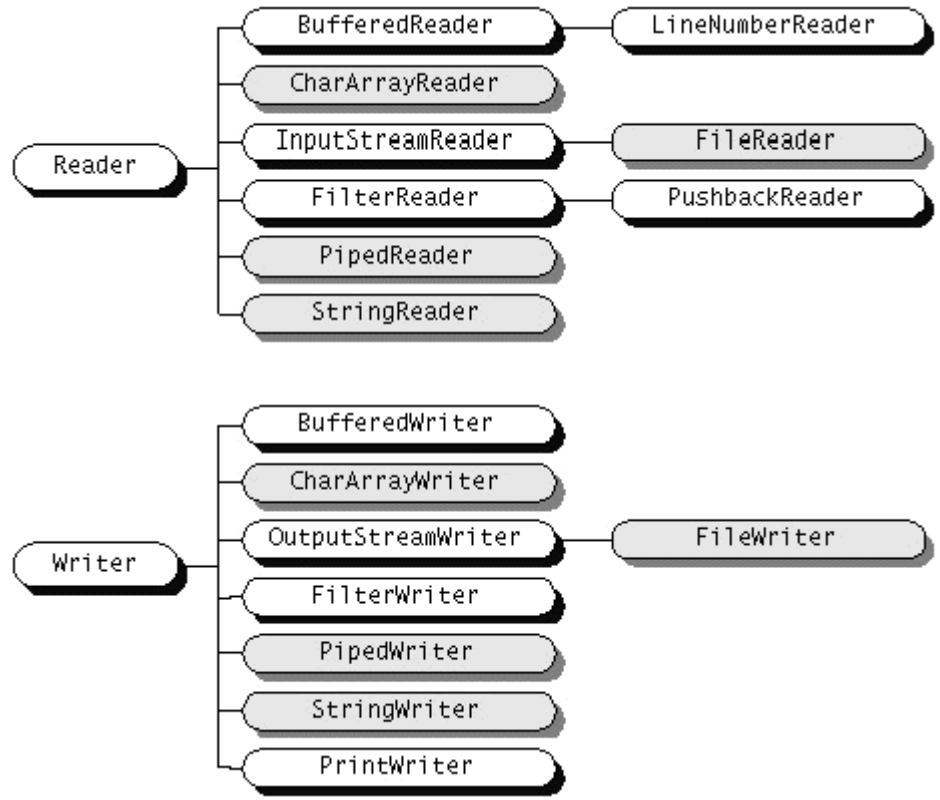
- Pachetul java.io defineste 2 tipuri de fluxuri:
  - Fluxuri orientate pe byte – scriere si citire pe 8 bit
    - Clasele InputStream si OutputStream
    - Citirea si scrierea de date binare (ex. imagine, sunet, video)
  - Fluxuri orientate pe caracter – scriere si citire pe 16 bit
    - Clasele Reader si Writer
    - Citirea si scrierea de caractere

Introducere

- Etapele in lucrul cu fluxuri:
  - *Deschiderea fluxului*
  - *Scriere / citire*
  - *Inchiderea fluxului*
- Inchiderea fluxurilor este necesara pentru ca acestea sunt resurse costisitoare si limitate
- Lucrul cu fluxuri implica lucrul cu exceptii (IOException)

Pachetul java.io





Accesarea fisierelor

- Se utilizeaza clasele: FileReader si FileWriter respective FileInputStreamReader si FileOutputStream

```
File file = new File("Hello1.txt");
 // creates the file
 file.createNewFile();
 // creates a FileWriter Object
 FileWriter writer = new FileWriter(file);
 // Writes the content to the file
 writer.write("This\n is\n an\n example\n");
 writer.flush();
 writer.close();
```

```
//Creates a FileReader Object
FileReader fr = new FileReader(file);
char [] a = new char[50];
fr.read(a); // reads the content to the array
for(char c : a)
   System.out.print(c); //prints the characters one by one
fr.close();
```

Utilizarea fluxurilor de tip Buffered

- Scrierea si citirea este mai eficienta
- Se utilizeaza un buffer intermediar
- Fortarea golirii buffe-rului cu flush()

```
BufferedReader br = null;

try {

        String sCurrentLine;

        br = new BufferedReader(new FileReader("C:\\testing.txt"));

        while ((sCurrentLine = br.readLine()) != null) {
                System.out.println(sCurrentLine);
        }

} catch (IOException e) {
        e.printStackTrace();
} finally {
        try {
                if (br != null)br.close();
        } catch (IOException ex) {
                ex.printStackTrace();
        }
}
```

Utilizarea fluxurilor buffered

- Versiunea 1.7 Java
- *“try with resources”*

```
try (BufferedReader br = new BufferedReader(new FileReader("C:\\testing.txt")))
                        {

                                String sCurrentLine;

                                while ((sCurrentLine = br.readLine()) != null) {
                                        System.out.println(sCurrentLine);
                                }

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

Conversia fluxurilor

- Converisa fluxurilor byte <-> character se face cu clasele:
  - InputStreamReader
  - OutputStreamWriter
- Exemplu:

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 System.out.print("Enter String");
 String s = br.readLine();
```

Fluxuri de tip pipe

- Utilizate pentru comunicarea intre fire de executie (Threads) – transmitere de date intre component ale aceluias program
- Clasele utilizate: PipeReader si PipeWriter / PipiedInputStream si PipedOutputStream

```
try{
        PipedReader in = new PipedReader();
        PipedWriter out = new PipedWriter();
        in.connect(out);

        //scrie date in pipe
            out.write("mesaj scris in pipe");

        //citeste din pipe
        while(in.ready()){
            int x = in.read();
            System.out.println("Read from pipe:"+(char)x);
        }

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

Fluxurile standard IO

- Trei steram-uri standard gestionate de clasa *java.lang.System*
- Intrarea standard – referita de **System.in**
  - utilizata pentru citirea de la tastatura
- Iesirea standard – referita de **System.out**
  - utilizata pentru afisarea de mesaje pe ecran
- Iesirea de eroare – referita de **System.err**
  - Utilizata pentru afiseare de mesaje de eroare (nu este folosit buffer)
- Aceste stream-uri pot fi redirectionate:

```
PrintStream ps = new PrintStream("C:/sample.txt");
System.setOut(ps);
```

Serializarea obiectelor

- Serializarea obiectelor = procesul de transmitere a unui obiect printr-un stream catre o destinatie utilizand in flux de iesire de tip *ObjectOutputStream*
- Deserializarea obiectelor = procesul de citire a unui obiect dintr-un flux de tip *ObjectInputStream* si incarcare in memorie
- Mecanism utilizat in cadrul Remote Method Invocation (RMI)

Serializarea obiectelor

- Clasele serializabile trebuie sa implementeze interfata *Serializable* sau *Externalizable*
- Cuvantul cheie *transient* blocheaza serializarea unui atribut

Serializarea obiectelor - Exemplu

```
class Alien implements Serializable{
    String name;
    transient int id;

    public Alien(String n) {
        this.name = n;
        id =  (int)(Math.random()*100);
    }
    public void move(){
         System.out.println("Alien is moving."+this);
     }
    public String toString(){
            return "[alien="+name+":id="+id+"]";}
}//.class
```

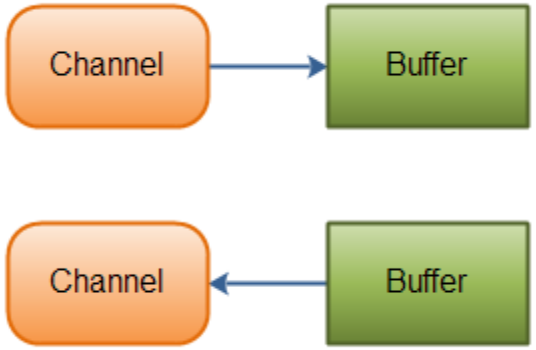
Serializarea obiectelor - Exemplu

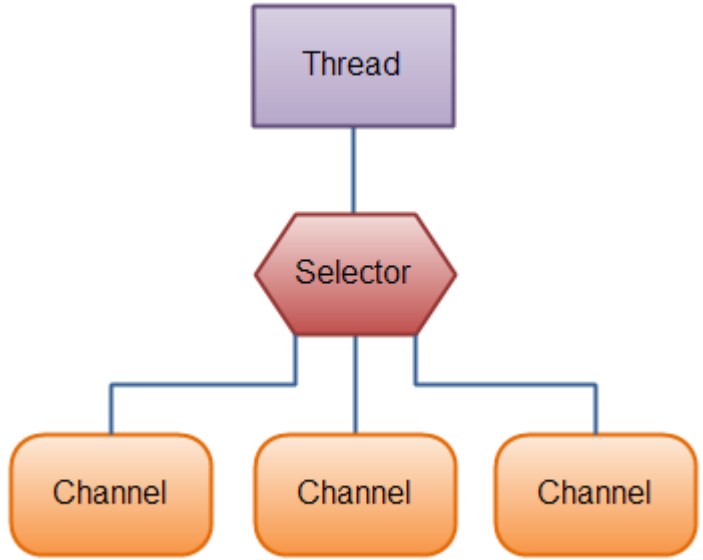
```
//...serializare obiect
  void freezAlien(Alien a, String storeRecipientName) throws
IOException{
        ObjectOutputStream o =
          new ObjectOutputStream(
           new FileOutputStream(storeRecipientName));

        o.writeObject(a);
        System.out.println(a+":I'll be back.");
    }
//...
```

```
//...deserializare obiect
  Alien unfreezAlien(String storeRecipientName) throws
IOException, ClassNotFoundException{
        ObjectInputStream in =
             new ObjectInputStream(
              new FileInputStream(storeRecipientName));
        Alien x = (Alien)in.readObject();
        System.out.println(x+":I'm back.");
        return x;
    }
//...
```

Java 5 – NIO

- *Java NIO* – NIO ofera o metoda alternativa de lucru cu fluxurile IO
- Concepte introduse:
  - Non-blocking IO – fluxuri neblocante
  - Channel si Buffer – datale sunt citite de canale in buffere
  - Selector – monitorizare canale multiple





Java 5 - NIO

```
RandomAccessFile aFile = new RandomAccessFile("data/nio-data.txt", "rw");
FileChannel inChannel = aFile.getChannel();

//create buffer with capacity of 48 bytes
ByteBuffer buf = ByteBuffer.allocate(48);

int bytesRead = inChannel.read(buf); //read into buffer.
while (bytesRead != -1) {

  buf.flip();  //make buffer ready for read

  while(buf.hasRemaining()){
      System.out.print((char) buf.get()); // read 1 byte at a time
  }

  buf.clear(); //make buffer ready for writing
  bytesRead = inChannel.read(buf);
}
aFile.close();
```

Java 7 – NIO2

- NIO2 si NIO nu sunt sinonime – NIO2 ofera o metoda alternative pentru lucrul cu fisiere
- Clase principale
  - *Paths* si *Path* - utilizate pentru referirea locatiilor fisierelor si directoarelor
  - *Files* – utilizata pentru manipularea continutului fisierelor
    - Metoda *File.toPath* permite interoperarea codului vechi cu java.nio

Java 7 – NIO2

```
byte[] readSmallBinaryFile(String aFileName) throws
IOException {
   Path path = Paths.get(aFileName);
   return Files.readAllBytes(path);
 }

void writeSmallBinaryFile(byte[] aBytes, String aFileName)
throws IOException {
   Path path = Paths.get(aFileName);
   Files.write(path, aBytes); //creates, overwrites
 }
```