Curs 6 PPOO

Conf. univ. dr. Cristian CIUREA

Departamentul de Informatică și Cibernetică Economică

cristian.ciurea@ie.ase.ro

Java fundamentals

- Fișiere – operații de I/O în Java
- Serializare

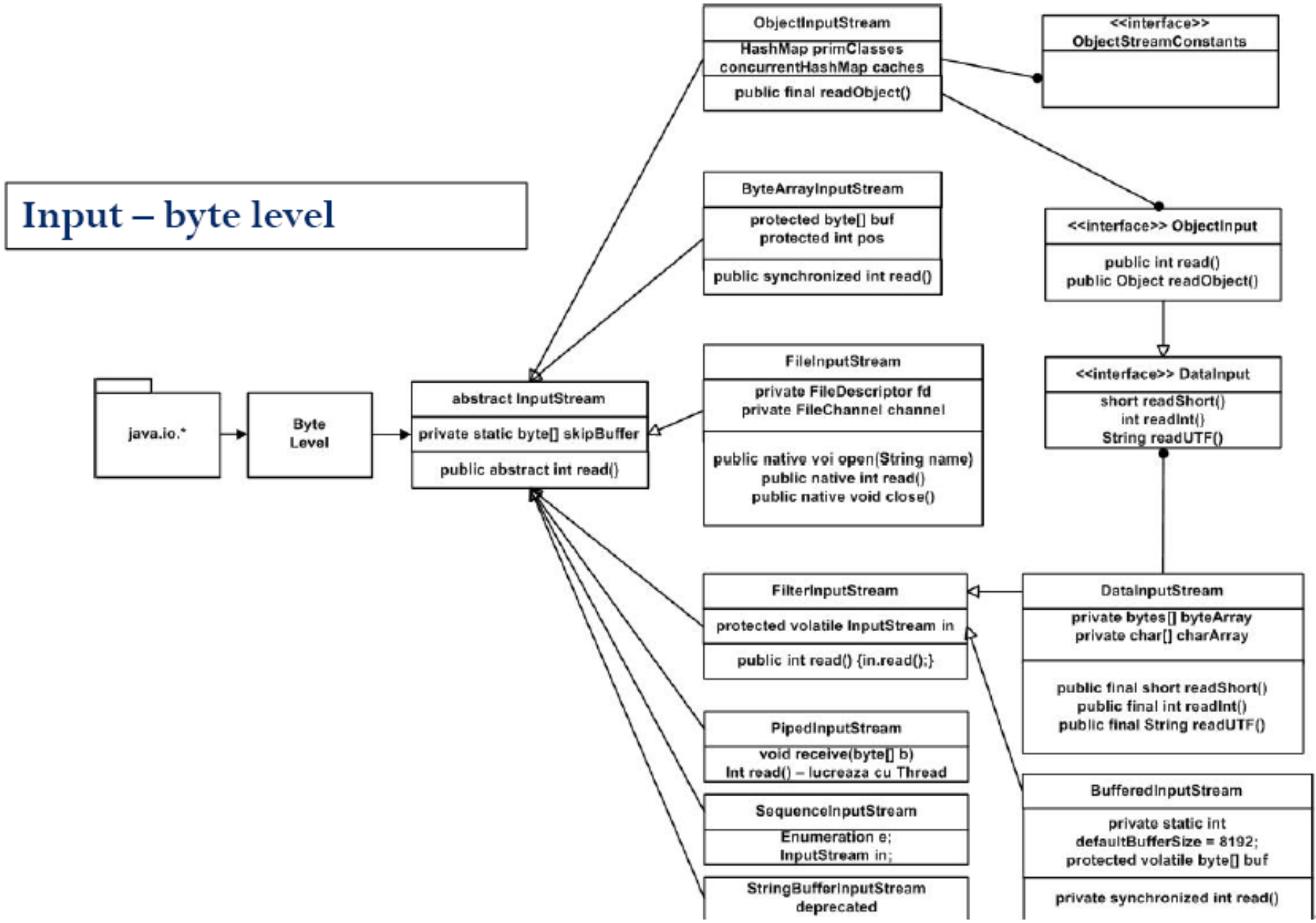
Fișiere

- Operațiile de intrare/ieșire sunt realizate, în general, cu ajutorul claselor din pachetul **java.io**.
- Bibliotecile I/O folosesc conceptul de **stream** (*flux*), ce reprezintă orice **sursă** sau **consumator** de date care este capabil să producă sau să primească **unități** de date, într-o manieră **secvențială**.

Fișiere

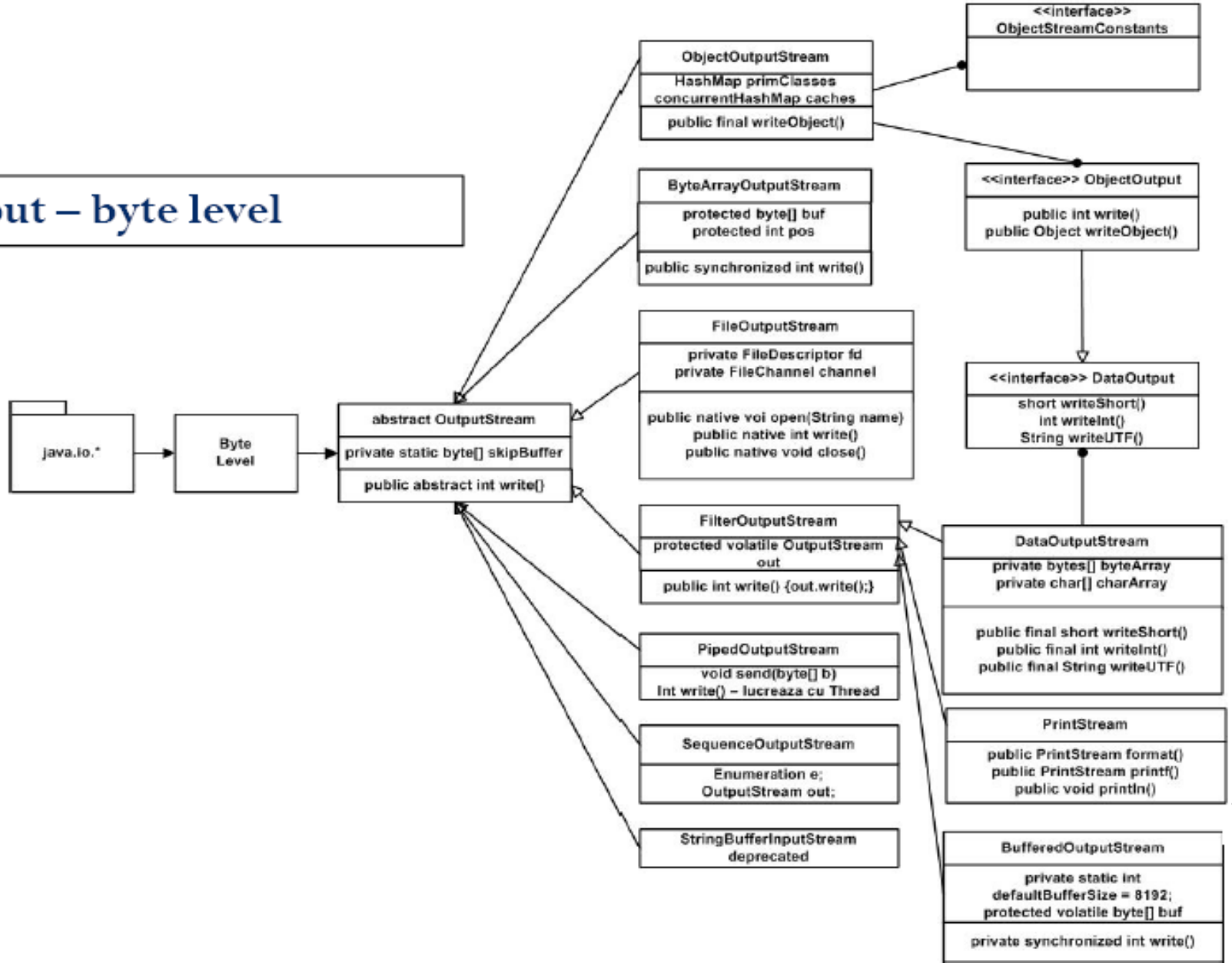
- Operațiile de I/O se bazează pe fluxuri de date:
  - la nivel de bait: **InputStream**, **OutputStream**;
  - la nivel de caracter (în Java un caracter = 2 B): **Reader**, **Writer**.
- Fișierele în Java sunt gestionate prin obiecte din clasa **File**.
- Majoritatea funcțiilor din aceste clase pot arunca excepții de tip **IOException** sau derivate din aceasta, cum este **FileNotFoundException**.

Fișiere

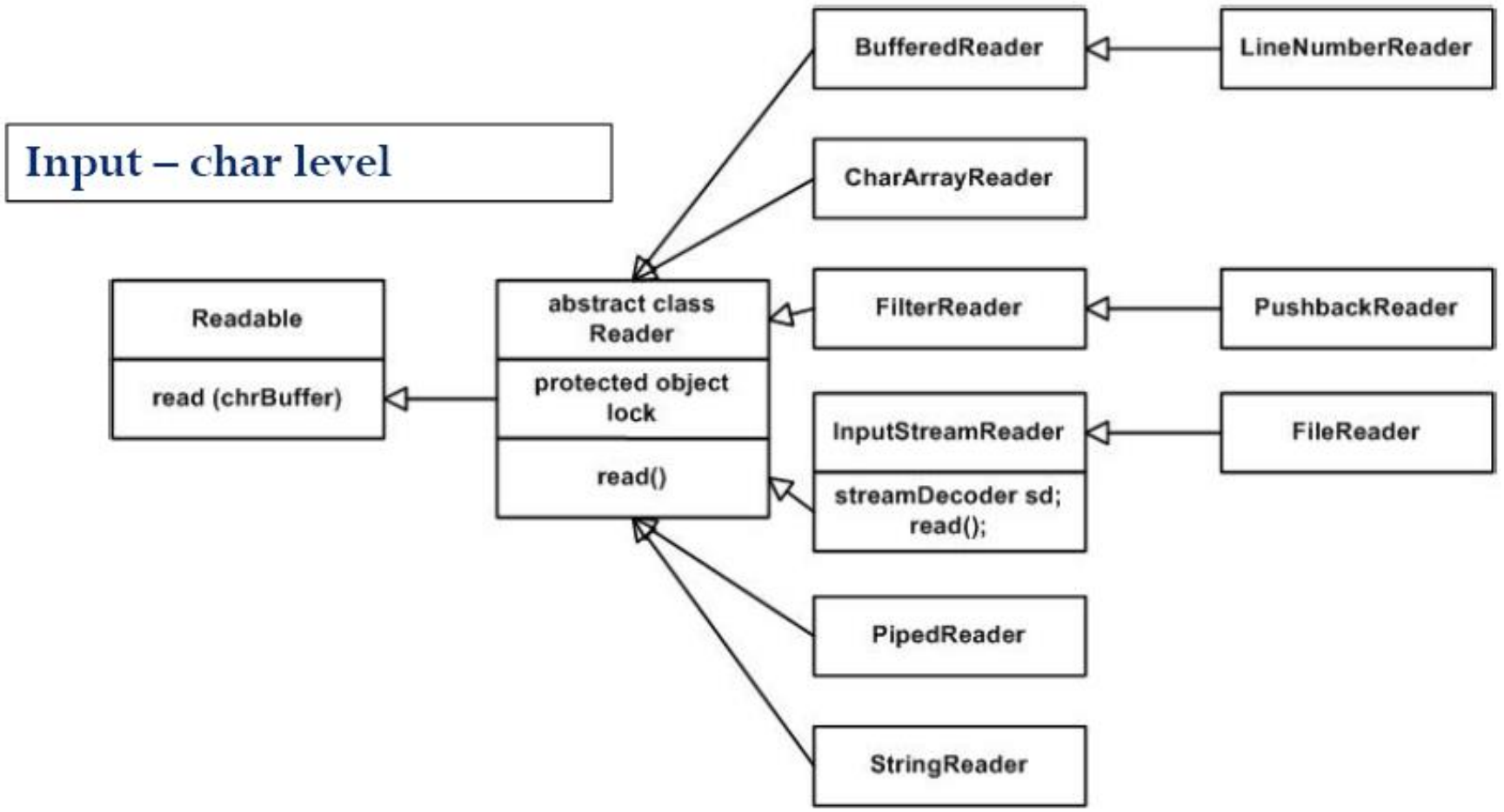

Input – byte level
java.io.*
Byte Level
abstract InputStream
private static byte[] skipBuffer
public abstract int read()
ObjectInputStream
HashMap primClasses
concurrentHashMap caches
public final readObject()
<<interface>> ObjectStreamConstants
ByteArrayInputStream
protected byte[] buf
protected int pos
public synchronized int read()
<<interface>> ObjectInput
public int read()
public Object readObject()
FileInputStream
private FileDescriptor fd
private FileChannel channel
public native voi open(String name)
public native int read()
public native void close()
<<interface>> DataInput
short readShort()
int readInt()
String readUTF()
FilterInputStream
protected volatile InputStream in
public int read() {in.read();}
DataInputStream
private bytes[] byteArray
private char[] charArray
public final short readShort()
public final int readInt()
public final String readUTF()
PipedInputStream
void receive(byte[] b)
Int read() – lucreaza cu Thread
SequenceInputStream
Enumeration e;
InputStream in;
StringBufferInputStream
deprecated
BufferedInputStream
private static int
defaultBufferSize = 8192;
protected volatile byte[] buf
private synchronized int read()

Fișiere

Output – byte level

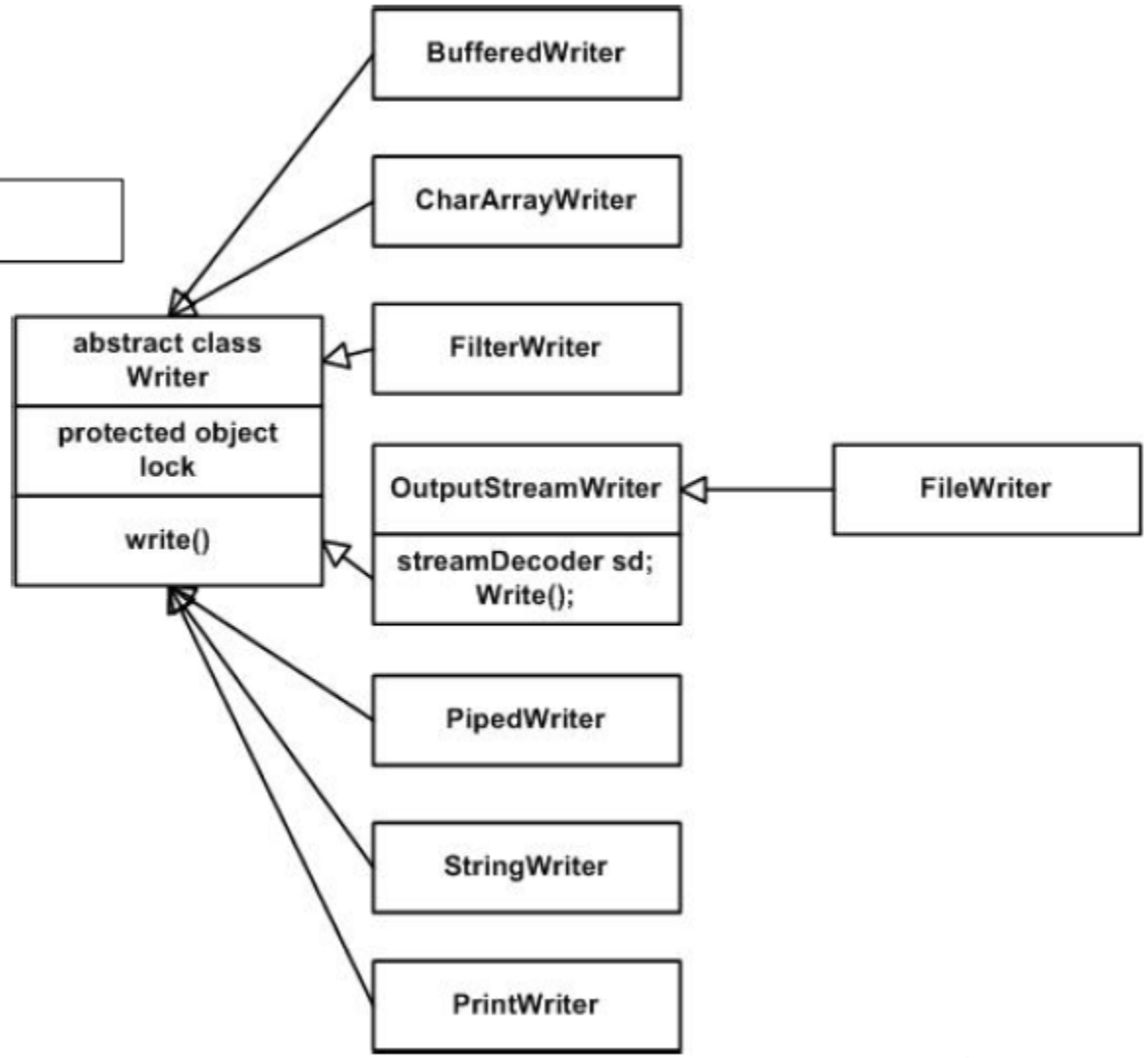


Fișiere


Input – char level
Readable
read (chrBuffer)
abstract class Reader
protected object lock
read()
BufferedReader
LineNumberReader
CharArrayReader
FilterReader
PushbackReader
InputStreamReader
FileReader
streamDecoder sd;
read();
PipedReader
StringReader

Fișiere

Output – char level



Fișiere

- **InputStreamReader, OutputStreamWriter** - convert bytes to characters and vice versa
- **DataInputStream, DataOutputStream** - read and write simple data types
- **ObjectInputStream, ObjectOutputStream** – read and write serialized Java objects
- **BufferedInputStream,BufferedOutputStream, BufferedReader, BufferedWriter** - stream filters with buffering

Fișiere

- **PrintStream**, **PrintWriter** – prints text;
- **FileInputStream**, **FileOutputStream** - implementations of InputStream, OutputStream
- **FileReader**, **FileWriter** - implementations of Reader, and Writer

Fișiere

- Există conversii de la șiruri de octeți la șiruri de caractere. Filtrul ce realizează această conversie este **InputStreamReader**, care obține perspectiva caracter asupra unui stream octet.
- **InputStreamReader** folosește o codificare dată ca parametru în constructor sau folosește codificarea implicită.
- O aplicație foarte importantă a acestui mecanism o reprezintă citirea de la consolă, ținând cont de faptul ca **System.in** are tipul **InputStream**. Deși era mai firesc ca tipul sa fie **Reader** (orientat caracter), acesta a rămas orientat octet din motive de compatibilitate: în versiunea inițială nu existau stream-uri caracter.

Fișiere

- Stream-urile au limitarea de a oferi o perspectivă exclusiv **secvențială** asupra unei surse/destinații de date.
- La un moment dat, nu putem sări direct la un octet/caracter din flux, ci trebuie să consumăm unitatea imediat următoare.
- Anumite stream-uri, cum ar fi pipe-urile sau socket-ii, nu pot fi abordate decât secvențial. Pentru fișiere situația stă altfel: ele reprezintă obiecte care au asociat un **cursor**, ce indică poziția următoarei operații de citire/scriere și care poate fi deplasat (***seek***).

Fișiere

Clasa **File** permite manipularea fișierelor ca entități, oferind o reprezentare abstractă a căilor de fișiere și directoare. Aceasta oferă operații precum:

- **`boolean createNewFile()`** – creează un nou fișier la calea specificată în constructor;
- **`boolean delete()`** – șterge fișierul;
- **`boolean exists()`** – verifică dacă fișierul/directorul dat de cale există;
- **`boolean isDirectory()`** – întoarce true atunci când calea denotă un director;
- **`boolean isFile()`** - întoarce true atunci când calea denotă un fisier;
- **`long length()`** – întoarce dimensiunea fisierului;
- **`String[] list()`** – întoarce intrările din director.

Fișiere

Class	Extends	Constructor	Methods
File	Object	(String) (String,String) (File,String)	createNewFile() delete() exists() isDirectory() isFile() list() mkdir() renameTo()
FileWriter	Writer	(File) (String)	cose() flush() write()
BufferedWriter	Writer	(Writer)	close() flush() newLine() write()

Fișiere

Class	Extends	Constructor	Methods
PrintWriter	Writer	(File) (String) (OutputStream) (Writer)	close() flush() write() print() println() format()
FileReader	Reader	(File) (String)	read()
BufferedReader	Reader	(Reader)	read() readLine()

Fișiere

Stream-urile standard, pe care le posedă orice program, sunt:

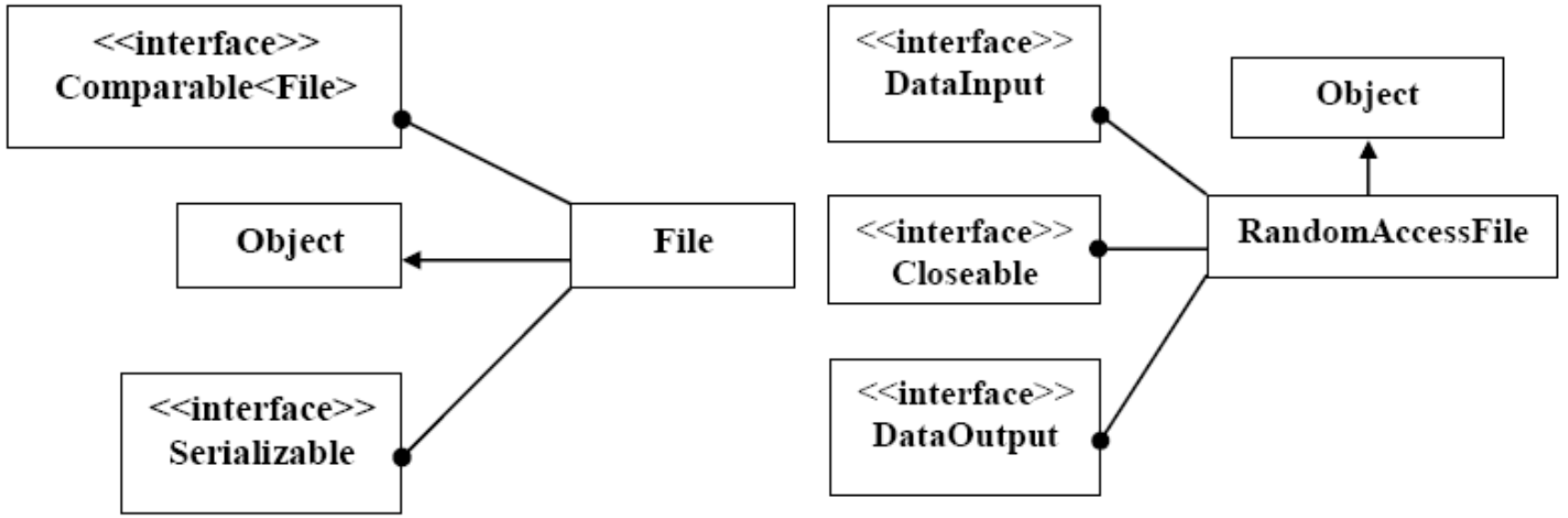
- **System.in**, de tip **InputStream;**
- **System.out**, de tip **PrintStream;**
- **System.err**, de tip **PrintStream**.

Clasa **PrintStream** oferă posibilitatea de formatare ca șir de caractere a valorilor având tipuri primitive. În general, nu este foarte folosită.

Fișiere

- Spre deosebire de clasele menționate anterior, care gestionează **conținutul** fișierelor, clasa **File** permite manipularea fișierelor ca **entități**, oferind o reprezentare abstractă a căilor de fișiere și directoare.
- Clasa **RandomAccessFile** nu aparține niciunei ierarhii de clase menționate anterior. Ea oferă posibilitatea deplasării prin fișierul deschis, folosind **seek** și implementează funcționalitatea **DataInput** și **DataOutput** (cea implementată și în **DataInputStream**, **DataOutputStream**), pentru a putea lucra cu tipuri primitive. Este asemănătoare ca funcționalitate cu folosirea fișierelor în ANSI C (fseek, fread, fwrite, etc).

Fișiere



File & RandomAccessFile

Serializare

- Serializarea presupune salvarea stării unui obiect;
- Deserializarea presupune încărcarea/restaurarea stării unui obiect;
- Operațiile sunt realizate prin apelul metodelor **ObjectInputStream.writeObject()**, respectiv **ObjectOutputStream.readObject()**;
- Clasa ale cărei obiecte se serializează trebuie să implementeze interfața **Serializable**;
- Atributele marcate ca **transient** nu sunt serializate.

Serializare

- you can override the mechanism:

```
private void writeObject(ObjectOutputStream os){
os.defaultWriteObject();
//other data
}
```

Serializare

- În cazul unei clase serializabile derivată dintr-o clasă non-serializabilă, toate variabilele instanță moștenite vor fi resetate la valorile primite în constructorul clasei derivate;
- Variabilele statice nu sunt serializate;
- Serializarea utilizează o valoare hash pe 64 biți, denumită **Serial Version UID (SUID)**, pentru a stoca versiunea structurii clasei:
- **static final long serialVersionUID = 1L;**

Serializare

- Fiecare clasă serializabilă are un număr unic de identificare asociat cu aceasta (**SUID**).
- În cazul în care numărul de identificare nu este specificat în mod explicit prin declararea unui câmp **private static final long**, denumit **serialVersionUID**, sistemul îl generează în mod automat, prin aplicarea unei proceduri deterministe complexe asupra clasei.
- În cazul în care clasa este modificată în vreun fel, câmpul **serialVersionUID** generat automat se modifică.

Serializare

- Cu excepția cazului în care programatorul dorește să definească explicit o schemă particulară de serializare, sistemul poate asigura serializarea automată a întregii componențe a obiectelor, explorând recursiv structura acestuia.
- Graful de referințe obținut poartă denumirea de **web of objects**.
- Serializarea este utilizată pentru implementarea conceptului de **persistență**, văzută drept capacitatea unui obiect de a supraviețui după încheierea execuției programului.

Bibliografie

- [1] Jonathan Knudsen, Patrick Niemeyer – *Learning Java, 3rd Edition*, O'Reilly.
- [2] http://www.itcsolutions.eu
- [3] http://www.acs.ase.ro
- [4] http://docs.oracle.com/javase/tutorial/ /index.html
- [5] http://cursuri.cs.pub.ro/~poo/wiki/index.php/Input/Output
- [6] http://cursuri.cs.pub.ro/~poo/wiki/index.php/Serializare