Advanced Programming Methods

Lecture 3 - Java IO Operations

Overview

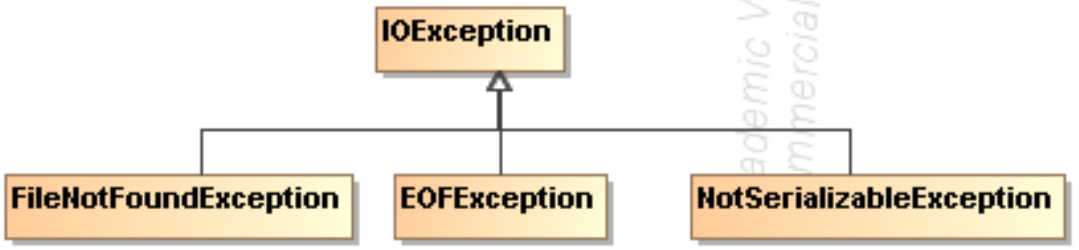
1. Java IO
2. Java NIO
3. Try-with-resources

Java IO

Java.IO

- Package java.io
  - classes working on bytes (InputStream, OutputStream)
  - Classes working on chars (Reader, Writer)
  - Byte-char conversion (InputStreamReader, OutputStreamWriter)
  - Random access (RandomAccessFile)
  - Scanner

- Exceptions:

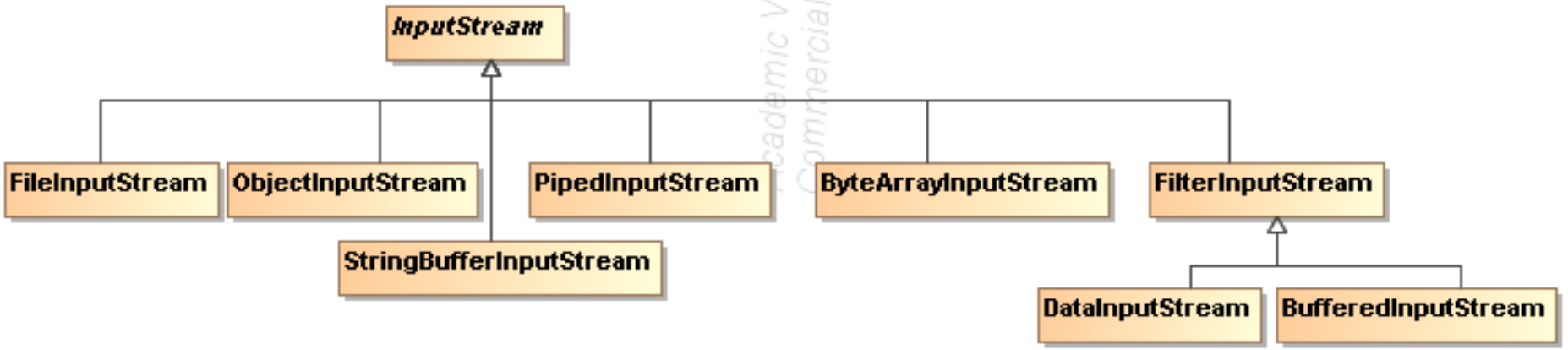


IO Stream

- represents an input source or an output destination
- is a sequence of data
- supports many different kinds of data, including simple bytes, primitive data types, localized characters, and objects.
- simply passes on data; others manipulate and transform the data in useful ways.

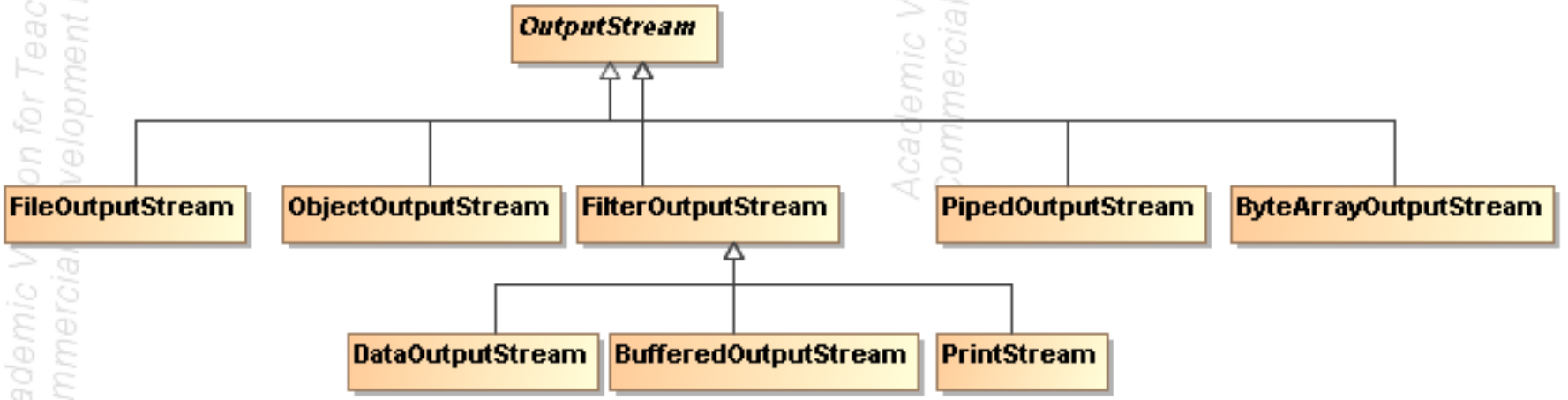
InputStream

- Abstract class that contains methods for reading bytes from a stream (file, memory, pipe, etc.)
  - `read():int` //read a byte, return –1 if no more bytes
  - `read(cbuff:byte[]): int` //read max `cbuff.length bytes`, return the nr of bytes that has been read, or -1
  - `read(buff:byte[], offset:int, length:int):int` //read max `length` bytes and write to `buff` starting with position `offset`, return the number of bytes that has been read, or -1
  - `available(): int` //number of bytes available for reading
  - `close()` //close the stream



OutputStream

- Abstract class that contains methods for writing bytes into a stream (file, memory, pipe, etc.)
  - `write(int)` //write a byte
  - `write(b:byte[])` //write `b.length` bytes from array `b` into the stream
  - `write(b:byte[], offset:int, len:int):int` //write `len` bytes from array `b` starting with the position `offset`
  - `flush()` // force the effective writing into the stream
  - `close()` //close the stream

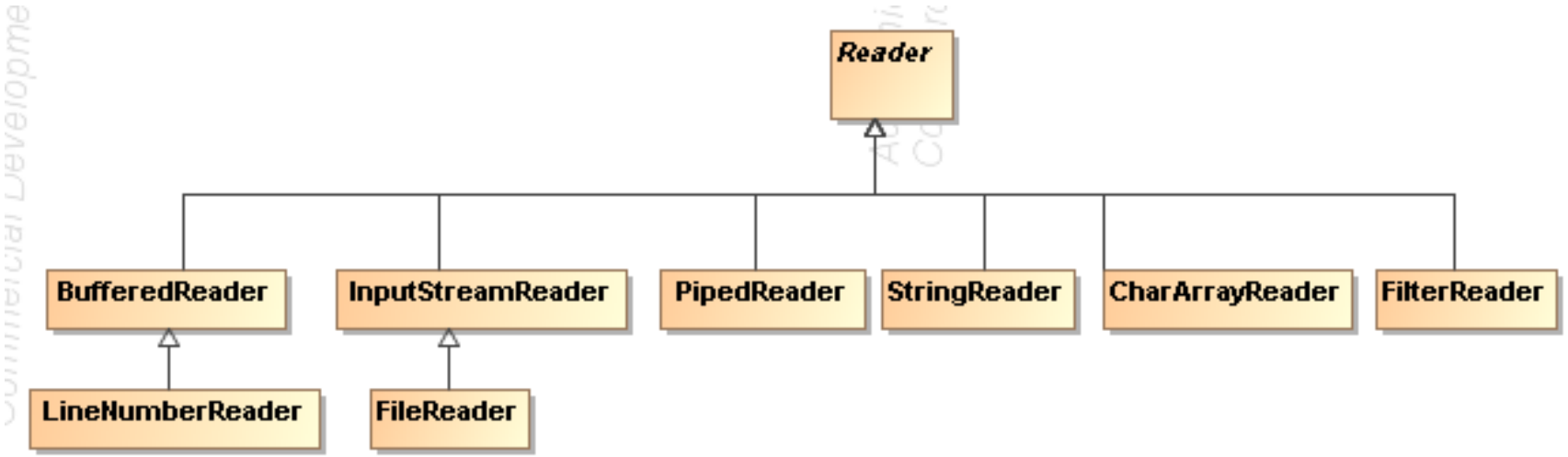


Example

```
FileInputStream in = null;
FileOutputStream out = null;
try {
   in = new FileInputStream(“fisier.txt");
   out = new FileOutputStream(“fisier2.txt");
   int c;
   while ((c = in.read()) != -1) {
          out.write(c);
   }
 } catch(IOException e){
    System.err.println(“Eroare ”+e);
 }finally {
   if (in != null)
      try {
          in.close();
      } catch (IOException e){ System.err.println("eroare "+e);}
   if (out != null)
       try {
           out.close();
       } catch (IOException e) { System.err.println("eroare "+e);}
}
```

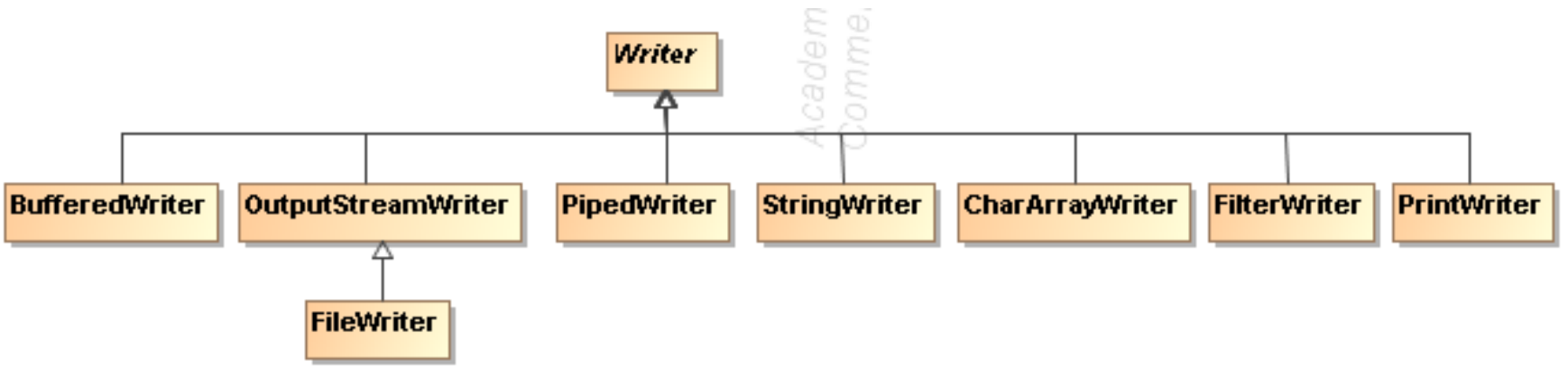
Reader

- Abstract class that contains methods for chars reading (1 char = 2 bytes) from a stream (file, memory, pipe, etc.)
  - `read():int` //read a char, return –1 for the end of the stream
  - `read(cbuff:char[]): int` //read max `cbuff.length` chars, return nr of read chars or -1
  - `read(buff:char[], offset:int, length:int):int` //read max `length` chars into array `buff` starting with offset `offset`, return the number of read chars or -1
  - `close()` //close the stream



Writer

- Abstract class that contains the methods for writing chars into a stream
  - `write(int)` //write a char
  - `write(b:char[])` //write `b.length` chars from array `b` into the stream
  - `write(b:char[], offset:int, len:int):int` //write `len` chars from array `b` starting with offset
  - `write(s:String)` //write a `String`
  - `write(s:String, off:int, len:int)` //write a part of a `String`
  - `flush()` // force the effective writing
  - `close()` //close the stream



Example

```
FileReader input = null;
FileWriter output = null;
try {
   input = new FileReader("Fisier.txt");
   output = new FileWriter(“Fisier2out.txt");
   int c;
   while ((c = input.read()) != -1) output.write(c);
} catch (IOException e) {
   System.err.println("Eroare la citire/scriere"+e);
} finally {
    if (input != null)
        try {
           input.close();
        } catch (IOException e) {System.err.println("eroare "+e);}
    if (output != null)
         try {
             output.close();
         } catch (IOException e) {System.err.println("Eroare "+e);}
}
```

Classes

Operations	**Byte**	**Char**
Files	FileInputStream, FileOutputStream	FileReader, FileWriter
Memory	ByteArrayInputStream, ByteArrayOutputStream	CharArrayReader CharArrayWriter
Buffered Operations	BufferedInputStream BufferedOutputStream	BufferedReader BufferedWriter
Format	PrintStream	PrintWriter
Conversion Byte ↔ Char	InputStreamReader (byte -> char) OutputStreamWriter (char -> byte)	

Example

- Read a list of students (from a file, from keyboard)
- Saving the list of students in ascending order based on their average (into a file)

//Studenti.txt

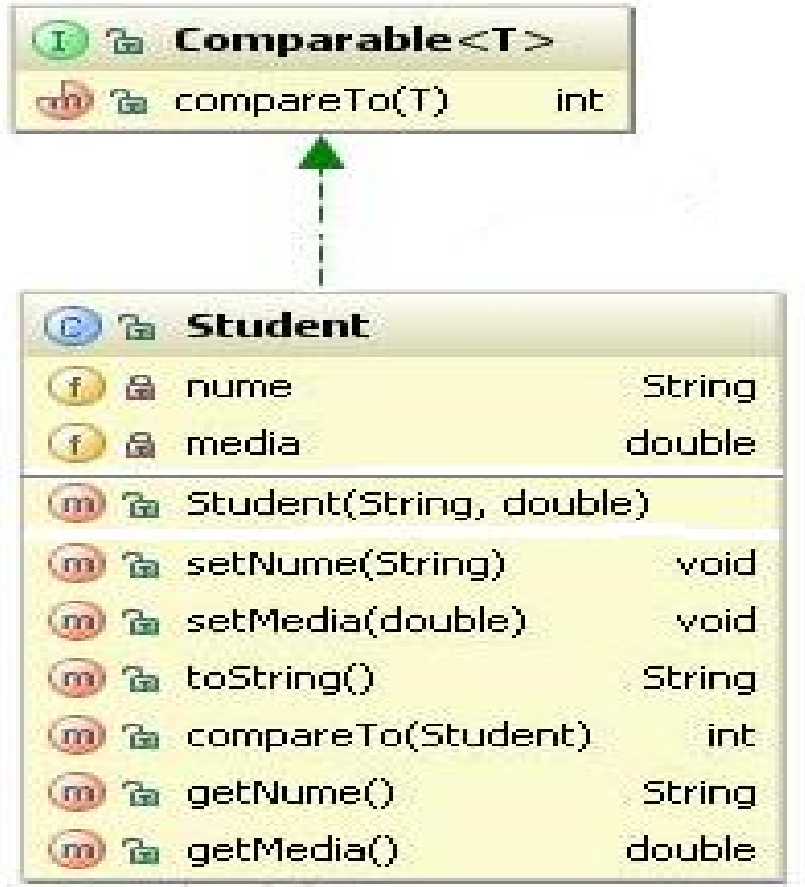
Vasilescu Maria|8.9

Popescu Ion|6.7

Marinescu Ana|9.6

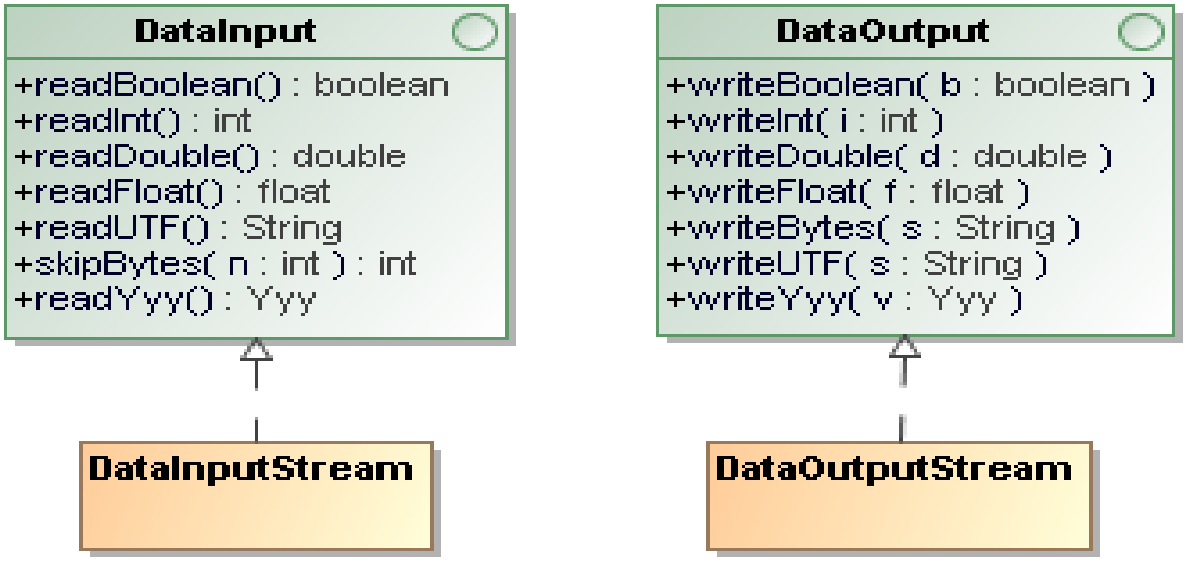
Ionescu George|7.53

Pop Vasile|9.3



Writing and reading data of primitive types

- Interfaces `DataInput` and `DataOutput`



- `Yyy` can be primitive types (`byte, short, char,` ...): `readByte(), readShort(), readChar(), writeByte(byte), writeShort(short), ...`

Obs:

1. In order to read data of primitive types using methods `readYyy,` the data must be saved before using the methods `writeYyy.`
2. When there are no more data it throws the exception `EOFException.`

Example DataOutput

```
void printStudentiDataOutput(List<Student> studs, String numefis){
        DataOutputStream output=null;
        try{
            output=new DataOutputStream(new FileOutputStream(numefis));
            for(Student stud: studs){
                output.writeUTF(stud.getNume());
                output.writeDouble(stud.getMedia());
            }
        } catch (FileNotFoundException e) {
            System.err.println("Eroare scriere DO "+e);
        } catch (IOException e) {
            System.err.println("Eroare scriere DO "+e);
        }finally {
            if (output!=null)
                try {
                    output.close();
                } catch (IOException e) {
                    System.err.println("Eroare scriere DO "+e);
                }
        }
}
```

Example DataInput

```
List<Student> citesteStudentiDataInput(String numefis){
   List<Student> studs=new ArrayList<Student>();
   DataInputStream input=null;
   try{
      input=new DataInputStream(new FileInputStream(numefis));
      while(true){
          String nume=input.readUTF();
          double media=input.readDouble();
          studs.add(new Student(nume,media));
      }
   }catch(EOFException e){ }
   catch (FileNotFoundException e) { System.err.println("Eroare citire"+e);}
   catch (IOException e) { System.err.println("Eroare citire DI "+e);}
   finally {
       if (input!=null)
         try { input.close();}
         catch (IOException e) {
                   System.err.println("Eroare inchidere fisier "+e);
          }}
   return studs;
}
```

Standard streams

- `System.in` of type `InputStream`
- `System.out` of type `PrintStream`
- `System.err` of type `PrintStream`

The associated streams can be modified using the methods:

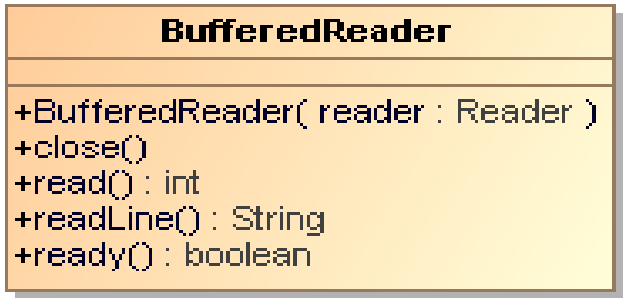
`System.setIn(), System.setOut(), System.setErr(),`

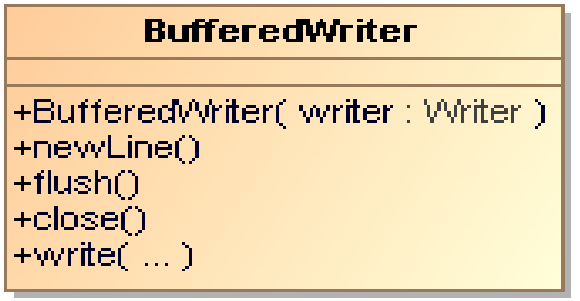
Example:

```
System.setOut(new PrintStream(new File("Output.txt")));
System.setErr(new PrintStream(new File("Erori.txt")));
```

BufferedReader/BufferedWriter

- Use a buffer to keep the data which are going to be read/write from/to a stream.
- Read/Write operations are more efficient since the reading/writing is effectively done only when the buffer is empty/full.






```
BufferedReader br=new BufferedReader(new FileReader(numefisier));
BufferedWriter bw=new BufferedWriter(new FileWriter(numefisier));
// rewrite the existing data in the file

//add at the end of the file
BufferedWriter bw=new BufferedWriter(new FileWriter(numefisier, true));
```

Example BufferedReader

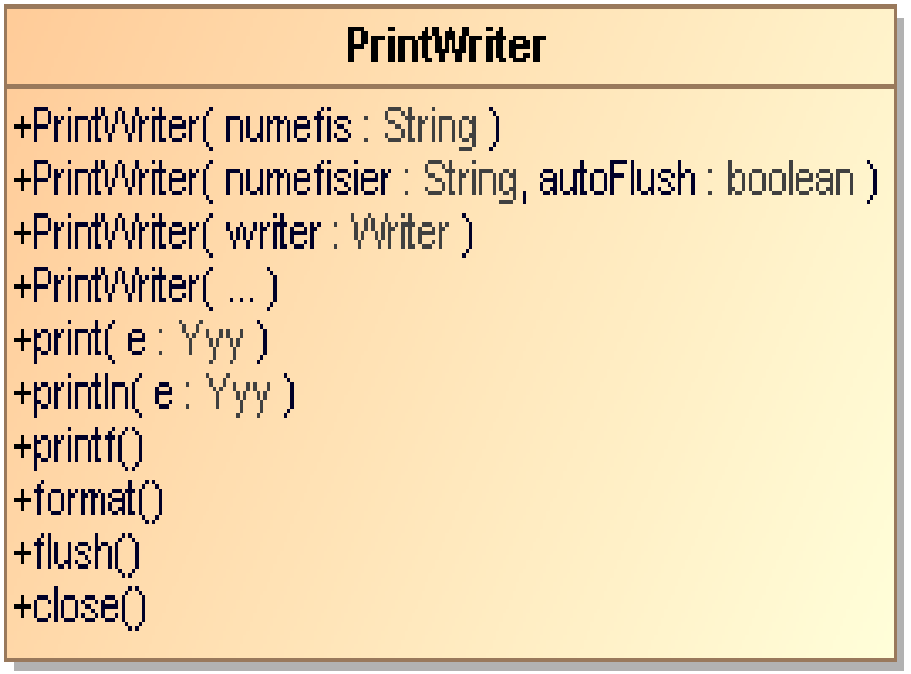
```
List<Student> citesteStudenti(String numefis){
   List<Student> ls=new ArrayList<Student>();
   BufferedReader br=null;
   try{
       br=new BufferedReader(new FileReader(numefis));
       String linie;
       while((linie=br.readLine())!=null){
           String[] elems=linie.split("[|]");
           if (elems.length<2){
               System.err.println("Linie invalida "+linie);
               continue;}
         Student stud=new Student(elems[0], Double.parseDouble(elems[1]));
         ls.add(stud);
         }
    }catch (FileNotFoundException e) {System.err.println("Eroare citire "+e);}
     catch (IOException e) { System.err.println("Eroare citire "+e);}
     finally{
        if (br!=null)
            try { br.close();}
            catch (IOException e) { System.err.println("Eroare inchidere fisier:
  "+e); }
    }
   return ls;
}
```

Example BufferedWriter

```
void printStudentiBW(List<Student> studs, String numefis){
        BufferedWriter bw=null;
        try{
            bw=new BufferedWriter(new FileWriter(numefis));
            //bw=new BufferedWriter(new FileWriter(numefis,true));
            for(Student stud: studs){
                bw.write(stud.getNume()+'|'+stud.getMedia());
                bw.newLine();    //scrie sfarsitul de linie
            }
        } catch (IOException e) {
            System.err.println("Eroare scriere BW "+e);
        } finally {
            if (bw!=null)
                try {
                    bw.close();
                } catch (IOException e) {
                    System.err.println("Eroare inchidere fisier "+e);
                }
        }
    }
```

PrintWriter

- Contains methods to save any type of data in text format.
- Contains methods to format the data .



- `Yyy` is any primitive or reference type. If `Yyy` is a reference type it is called the method `toString` corresponding to `e`.

Example 1 PrintWriter

```
void printStudentiPrintWriter(List<Student> studs, String numefis){
        PrintWriter pw=null;
        try{
            pw=new PrintWriter(numefis);
            for(Student stud: studs){
                pw.println(stud.getNume()+'|'+stud.getMedia());
            }

        } catch (FileNotFoundException e) {
            System.err.println("Eroare scriere PW "+e);
        }finally {
            if (pw!=null)
                pw.close();
        }
}
```

Example 2 PrintWriter

```
void printStudentiPWTabel(List<Student> studs, String numefis){
   PrintWriter pw=null;
   try{
      pw=new PrintWriter(numefis);
      String linie=getLinie('-',48);
      int crt=0;
      for(Student stud:studs){
         pw.println(linie);
//pw.printf("| %3d | %-30s | %5.2f |%n",(++crt),stud.getNume(),stud.getMedia());
  pw.format("| %3d | %-30s | %5.2f |%n",(++crt),stud.getNume(),stud.getMedia());
       }
       if (crt>0)
          pw.println(linie);
    } catch (FileNotFoundException e) {
          System.err.println("Eroare scriere PWTabel "+e);
    } finally {
          if (pw!=null)
              pw.close();
    }
 }
```

Example 2 PrintWriter

```
String getLinie(char c, int length){
        char[] tmp=new char[length];
        Arrays.fill(tmp,c);
        return String.valueOf(tmp);
}
//file result
-------------------------------------------
|   1 | Popescu Ion                 |  6.70 |
-------------------------------------------
|   2 | Ionescu George              |  7.53 |
-------------------------------------------
|   3 | Vasilescu Maria             |  8.90 |
-------------------------------------------
|   4 | Pop Vasile                  |  9.30 |
-------------------------------------------
|   5 | Marinescu Ana               |  9.60 |
-------------------------------------------
```

Reading from the keyboard

- Class `BufferedReader`

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

- Class `Scanner` (package `java.util`)

```
Scanner input=new Scanner(System.in);
```

- Class `Scanner` contains methods to read data of primitive types from the keyboard (or other stream):
  - `nextInt():int`
  - `nextDouble():double`
  - `nextFloat():Float`
  - `nextLine():String`
  - `...`
  - `hasNextInt():boolean`
  - `hasNextDouble():boolean`
  - `hasNextFloat():boolean`
  - `...`

Example BufferedReader (keyboard)

```
List<Student> citesteStudenti(){   //from the keyboard
  List<Student> ls=new ArrayList<Student>();
  BufferedReader br=null;
  try{
    br=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("La terminare introduceti cuvantul \"gata\"");
    boolean gata=false;
    while(!gata){
            System.out.println("Introduceti numele: ");
            String snume=br.readLine();
            if ("gata".equalsIgnoreCase(snume)){gata=true; continue;}
            System.out.println("Introduceti media: ");
            String smedia=br.readLine();
            if ("gata".equalsIgnoreCase(smedia)){gata=true; continue;}
            try{
               double media=Double.parseDouble(smedia);
               lista.add(new Student(snume,media));
            }catch(NumberFormatException nfe){
               System.err.println("Eroare: "+nfe);
            }
     } }/*catch, finally, ...*/ }//citesteStudenti
```

Example Scanner (keyboard)

```
List<Student> citesteStudentiScanner(){
  List<Student> lista=new ArrayList<Student>();
  Scanner scanner=null;
  try{
    scanner=new Scanner(System.in);
   System.out.println("La terminare introduceti cuvantul \"gata\"");
    boolean gata=false;
    while(!gata){
       System.out.println("Introduceti numele: ");
       String snume=scanner.nextLine();
       if ("gata".equalsIgnoreCase(snume)){gata=true; continue;  }
       System.out.println("Introduceti media: ");
       if (scanner.hasNextDouble()) {
           double media=scanner.nextDouble();
           lista.add(new Student(snume,media));
            scanner.nextLine();  //to read <Enter>
            continue;
        }else{
        //next slide
```

Example Scanner (keyboard) cont.

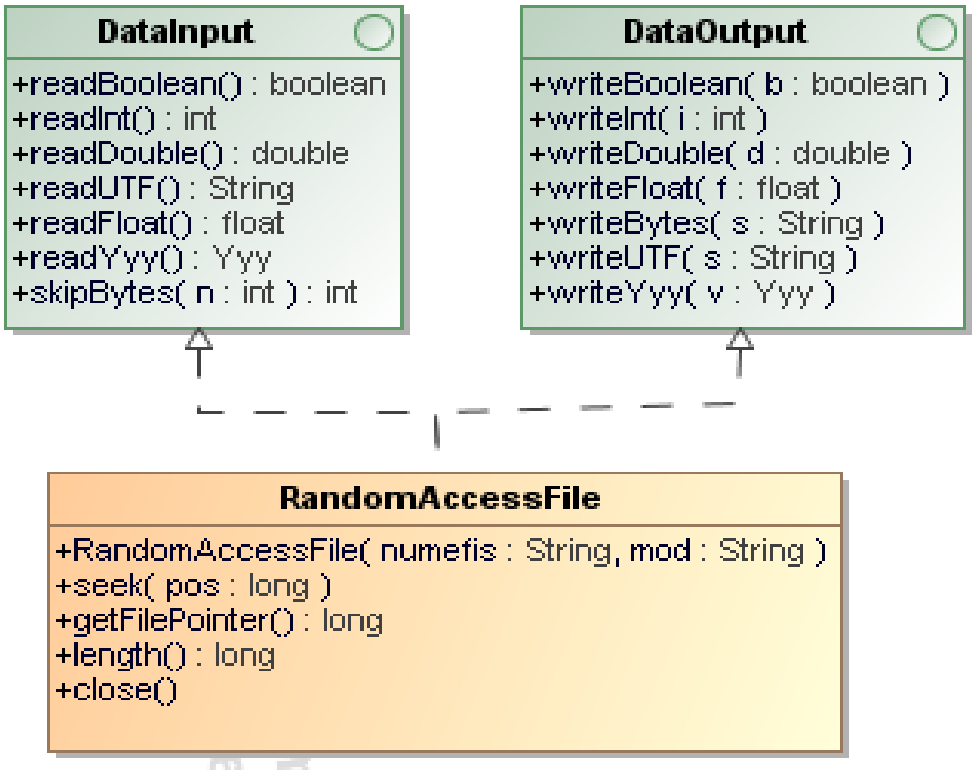
```
List<Student> citesteStudentiScanner(){
  //...
  if (scanner.hasNextDouble()){
    //...
  } else{
        String msj=scanner.nextLine();
        if ("gata".equalsIgnoreCase(msj)){
                        gata=true;
                        continue;
         }else
            System.out.println("Trebuie sa introduceti media studentului");
    }//else
   }//while
 } finally {
    if (scanner!=null)
         scanner.close();
 }
 return lista;
}
```

Example Scanner file

```
Scanner inscan=null;
try{
  inscan=new Scanner(new BufferedReader(new FileReader("intregi.txt")));
  //inscan.useDelimiter(",");
  while(inscan.hasNextInt()){
     int nr=inscan.nextInt();
     System.out.println("nr = " + nr);
  }
} catch (FileNotFoundException e) {
    System.err.println("Eroare "+e);
}finally {
   if (inscan!=null)
    inscan.close();
}
```

RandomAccessFile

- Allow random access to a file.
- Can be used for either reading or writing.
- Class uses the notion of cursor to denote the current position in the file. Initially the cursor is on the position 0 at the beginning of the file.
- Operations of reading/writing move the cursor according the number of bytes read/written.



Example writing RandomAccessFile

```
void printStudentiRAF(List<Student> studs, String numefis){
  RandomAccessFile out=null;
  try{
    out=new RandomAccessFile(numefis,"rw");
    for(Student stud: studs){
      out.writeUTF(stud.getNume());
      out.writeDouble(stud.getMedia());
    }
  }catch (FileNotFoundException e){System.err.println("Eroare RAF "+e);}
  catch (IOException e) { System.err.println("Eroare scriere RAF "+e);}
  finally{
     if (out!=null)
        try {
           out.close();
        } catch (IOException e) {
            System.err.println("Eroare inchidere fisier "+e);
        }
    }
 }
```

Example reading RandomAccessFile

```
List<Student> citesteStudentiRAF(String numefis){
  List<Student> studs=new ArrayList<Student>();
  RandomAccessFile in=null;
  try{
            in=new RandomAccessFile(numefis, "r");
            while(true){
                String nume=in.readUTF();
                double media=in.readDouble();
                studs.add(new Student(nume,media));
            }
  }catch(EOFException e){ }
  catch (FileNotFoundException e){System.err.println("Eroare la citire: "+e);}
  catch (IOException e) { System.err.println("Eroare la citire "+e);}
  finally {
      if (in!=null)
         try {
             in.close();
         } catch (IOException e) {System.err.println("Eroare RAF "+e);}
}
 return studs;     }
```

Example appending RandomAccessFile

```
void adaugaStudent(Student stud, String numefis){
        RandomAccessFile out=null;
        try{
            out=new RandomAccessFile(numefis, "rw");
            out.seek(out.length());
            out.writeUTF(stud.getNume());
            out.writeDouble(stud.getMedia());
        } catch (FileNotFoundException e) {
            System.err.println("Eroare RAF "+e);
        } catch (IOException e) {
            System.err.println("Eroare RAF "+e);
        }finally {
            if (out!=null)
                try {
                    out.close();
                } catch (IOException e) {
                    System.err.println("Eroare RAF "+e);
                }
        }
    }
```

Class File

- Represent the name of a file (not its content).
- Allow platform-independent operations on the files (create, delete, rename, etc.) .
  - `File(name:String) //name  is the path to a file or a directory`
  - `getName():String`
  - `getAbsolutePath():String`
  - `isFile():boolean`
  - `isDirectory():boolean`
  - `exists():boolean`
  - `delete():boolean`
  - `deleteOnExit()    //Directory/File is removed at the exit of JVM`
  - `mkdir()`
  - `list():String[]`
  - `list(filtru:FilenameFilter):String[]`
  - ...

Class File: examples

- Printing the current directory

```
File dirCurent=new File(".");
System.out.println("Directory:" + dirCurent.getAbsolutePath());
```

- Creating an OS-independent path

```
String namefis=".."+File.separator+"data"+File.separator+"intregi.txt";
File f1=new File(namefis);
System.out.println("F1 "+f1.getName());
System.out.println("Exists f1? "+f1.exists());
```

- Selecting the .txt files from a directory

```
File dir=new File(".");
String[] files=dir.list(new FilenameFilter(){
   public boolean accept(File dir, String name) {
      return name.toLowerCase().endsWith(".txt");
}  });
System.out.println("Files "+ Arrays.toString(files));
```

- Removing a file

```
File namef=new File("erori.txt");
if (namef.exists())
   boolean ok=namef.delete();
```

Java NIO

Java NIO (New IO)

- From Java 1.4
- is an alternative IO API for Java (to the standard Java IO and Java Networking API's)
- consist of the following core components:
  - Channels
  - Buffers
  - Selectors

Channels and Buffers

- In the standard IO API you work with byte streams and character streams.
- In NIO you work with channels and buffers. Data is always read from a channel into a buffer, or written from a buffer to a channel.
- all IO in NIO starts with a Channel.

Channels and Buffers

- Channels are similar to streams with a few differences:
  - You can both read and write to a Channels. Streams are typically one-way (read or write).
  - Channels can be read and written asynchronously.
  - Channels always read to, or write from, a Buffer.

Channels and Buffers

the most important Channel implementations in Java NIO:

- FileChannel: reads data from and to files.
- DatagramChannel: can read and write data over the network via UDP.
- SocketChannel: can read and write data over the network via TCP.
- ServerSocketChannel: allows you to listen for incoming TCP connections, like a web server does.

Channels and Buffers

- A buffer is essentially a block of memory into which you can write data, which you can then later read again.
- Buffer types let you work with the bytes in the buffer as char, short, int, long, float or double:
  - ByteBuffer
  - MappedByteBuffer
  - CharBuffer
  - DoubleBuffer
  - FloatBuffer
  - IntBuffer
  - LongBuffer
  - ShortBuffer

Channels and Buffers

- Using a Buffer to read and write data typically follows this little 4-step process:
  - Write data into the Buffer: The buffer keeps track of how much data you have written.
  - Call buffer.flip(): in order to switch the buffer from writing mode into reading mode
  - Read data out of the Buffer: In reading mode the buffer lets you read all the data written into the buffer.
  - Call buffer.clear() or buffer.compact(): to make buffer ready for writing again

Channels and Buffers

- The clear() method clears the whole buffer.
- The compact() method only clears the data which you have already read. Any unread data is moved to the beginning of the buffer, and data will now be written into the buffer after the unread data.

Channels and Buffers

```
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();
```

Channels and Buffers

- A Buffer has three properties:

  1. Capacity: a certain fixed size. Once the Buffer is full, you need to empty it (read the data, or clear it) before you can write more data into it.

  2. Position:

     - Write mode:Initially the position is 0. When a byte, long etc. has been written into the Buffer the position is advanced to point to the next cell in the buffer to insert data into. Position can maximally become capacity – 1
     - Read mode:When you flip a Buffer from writing mode to reading mode, the position is reset back to 0. As you read data from the Buffer you do so from position, and position is advanced to next position to read.

Channels and Buffers

3. Limit:

- Write mode: is the limit of how much data you can write into the buffer and it is equal to the capacity of the Buffer
- Read mode: is the limit of how much data you can read from the data. Therefore, when flipping a Buffer into read mode, limit is set to write position of the write mode. In other words, you can read as many bytes as were written (limit is set to the number of bytes written, which is marked by position).

Scattering Reads

- reads data from a single channel into multiple buffers

```
ByteBuffer buf1 = ByteBuffer.allocate(128);
ByteBuffer buf2   = ByteBuffer.allocate(1024);
ByteBuffer[] bufferArray = { buf1,buf2  };
channel.read(bufferArray);
```

Gathering Writes

- writes data from multiple buffers into a single channel

```
ByteBuffer buf1 = ByteBuffer.allocate(128);
ByteBuffer buf2   = ByteBuffer.allocate(1024);
ByteBuffer[] bufferArray = { buf1,buf2  };
channel.write(bufferArray);
```

Java NIO FileChannel

- writes data from multiple buffers into a single channel
- is a channel that is connected to a file.
- you can read data from a file, and write data to a file.
- is an alternative to reading files with the standard Java IO API.

Channel to Channel Transfer

- you can transfer data directly from one channel to another

```
RandomAccessFile fromFile = new RandomAccessFile("fromFile.txt", "rw");
FileChannel      fromChannel = fromFile.getChannel();

RandomAccessFile toFile = new RandomAccessFile("toFile.txt", "rw");
FileChannel      toChannel = toFile.getChannel();

long position = 0;
long count    = fromChannel.size();

toChannel.transferFrom(fromChannel, position, count);
```

Java NIO Path

- an interface that is similar to the java.io.File class
- An instance represents a path in the file system. A path can point to either a file or a directory. A path can be absolute or relative.

Java NIO Files

- provides several methods for manipulating files in the file system

Java NIO AsynchronousFileChannel

- makes it possible to read data from, and write data to files asynchronously
- Read and write operations can be done either via a Future or via a CompletionHandler

Reading via a Future

```
AsynchronousFileChannel fileChannel =
                AsynchronousFileChannel.open(path, StandardOpenOption.READ);


ByteBuffer buffer = ByteBuffer.allocate(1024);
long position = 0;


Future<Integer> operation = fileChannel.read(buffer, position);


while(!operation.isDone());


buffer.flip();
byte[] data = new byte[buffer.limit()];
buffer.get(data);
System.out.println(new String(data));
buffer.clear();
```

Writing via a CompletionHandler

```
Path path = Paths.get("data/test-write.txt");
if(!Files.exists(path)){
    Files.createFile(path);}
AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(path, StandardOpenOption.WRITE);
ByteBuffer buffer = ByteBuffer.allocate(1024);
long position = 0;
buffer.put("test data".getBytes());
buffer.flip();
fileChannel.write(buffer, position, buffer, new CompletionHandler<Integer, ByteBuffer>() {
    @Override
    public void completed(Integer result, ByteBuffer attachment) {
        System.out.println("bytes written: " + result);}
    @Override
    public void failed(Throwable exc, ByteBuffer attachment) {
        System.out.println("Write failed");
        exc.printStackTrace();}
});
```

Selectors

- A Selector allows a single thread to handle multiple Channel's.
- is handy if your application has many Channels open
- To use a Selector you register the Channel's with it. Then you call it's select() method. This method will block until there is an event ready for one of the registered channels. Once the method returns, the thread can then process these events. Examples of events are incoming connection, data received etc.

Try-with-resources

Old Style

```
private static void printFile() throws IOException {
    InputStream input = null;
    try {
        input = new FileInputStream("file.txt");
        int data = input.read();
        while(data != -1){
            System.out.print((char) data);
            data = input.read();
        }
    } finally {
        if(input != null){
            input.close();
        }
    }
}
```

- The red marked code may throw exceptions

Try-with-resources

- From Java 7 the previous code can be rewritten as follows:

```
private static void printFileJava7() throws IOException {

    try(FileInputStream input = new FileInputStream("file.txt")) {

        int data = input.read();
        while(data != -1){
            System.out.print((char) data);
            data = input.read();
        }
    }
}
```

Try-with-resources

- When the try block finishes the FileInputStream will be closed automatically. This is possible because FileInputStream implements the Java interface java.lang.AutoCloseable.

```
public interface AutoClosable {

    public void close() throws Exception;
}
```

- All classes implementing this interface can be used inside the try-with-resources construct.