**Programare Avansata pe Obiecte  
Laborator 7**

**Pahontu Bogdan-Ionut**

**E-mail:** [**pahontubogdan@gmail.com**](mailto:pahontubogdan@gmail.com)

Contents

[1. Working with files 3](#_Toc5657538)

[1.1. IO vs NIO 3](#_Toc5657539)

[1.2. Files class 3](#_Toc5657540)

[1.3. Streams 4](#_Toc5657541)

[1.4. Reader and Writer 6](#_Toc5657542)

[1.5. RandomAccessFile 7](#_Toc5657543)

[2. Tasks 8](#_Toc5657544)

# Working with files

## IO vs NIO

|  |  |
| --- | --- |
| IO | NIO |
| Stream oriented | Buffer oriented |
| Blocking IO | Non-blocking IO |

* The main differences between IO and NIO can be found at the following link: <https://howtodoinjava.com/java/io/difference-between-standard-io-and-nio/>
* After the appearance of NIO new classes such as Files and Paths were develop as utility classes:
  + Files – offers a lot of methods to work with files;
  + Paths – offers a lot of methods to work with files access routes;
  + Please consult the following links to see more info about this two classes:
    - <https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html>
    - <https://docs.oracle.com/javase/8/docs/api/java/nio/file/Paths.html>

## Files class

* This class is used in order to perform the following actions:
  + Read the metadata for a file or a folder;
  + Create or delete files or folders;
  + Has support for opening streams for reading or writing;
* An instance of this class will have a route path to the file or a folder pathname;
* This class CAN`T be used to read information from a file (here we will use streams);
* The following picture presents method that can be used for a Files objects;

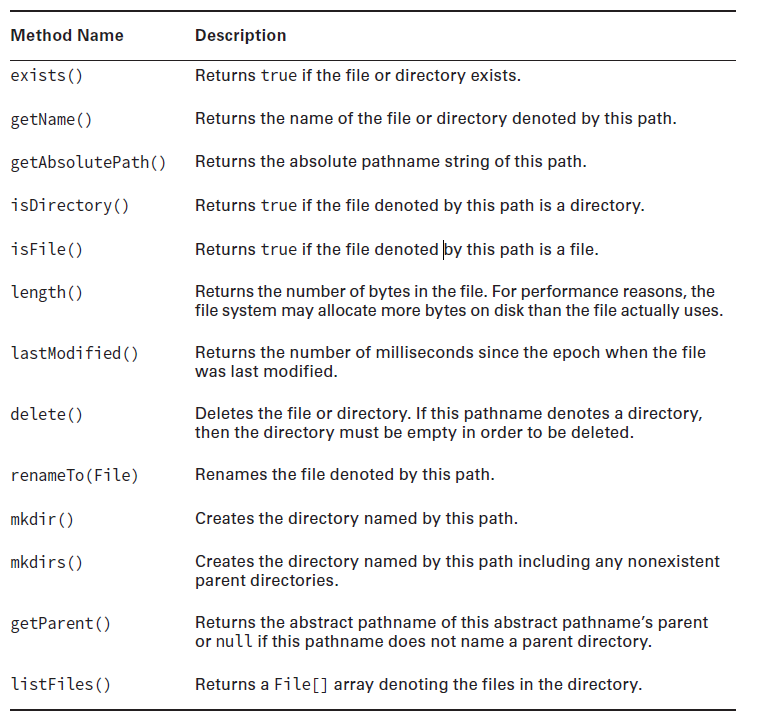


Figure 1 – Understanding exceptions

## Streams

* In Java, three steams are automatically created. These are initiated by Java runtime at the JVM starting:
  + System.out: standard output stream;

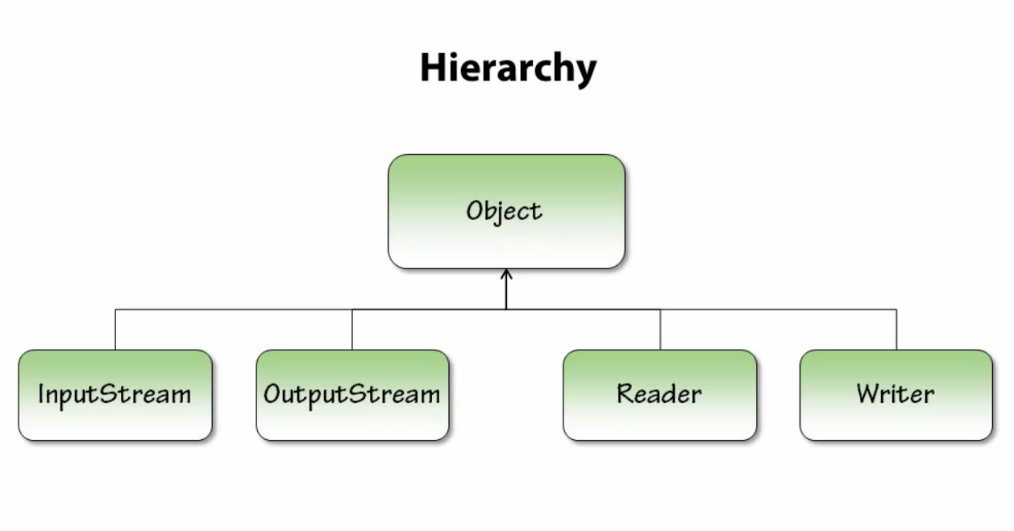
System.out.println("simple message");

* + System.in: standard input stream ;

int i=System.in.read(); //returns ASCII code of 1st character

System.out.println((char)i); //will print the character

* + System.err: standard error stream;



OutputStream (abstract)

ByteArrayOutputStream

FileOutputStream

FilterOutputStream

ObjectOutputStream

PipedOutputStream

BufferedOutputStream

DataOutputStream

PrintStream

Figure 2 – Output Streams

InputStream (abstract)

ByteArrayInputStream

FileInputStream

ObjectInputStream

PipedInputStream

SequenceInputStream

StringBufferInputStream

BufferedInputStream

DataInputStream

LineNumberInputStream

PushbackInputStream

Figure 3 – Input Streams

* The ones marked with green will be used in our laboratory and the ones with orange are deprecated;

## Reader and Writer

Writer (abstract)

BufferedWriter

CharArrayWriter

FilterWriter (abstract)

OutputStreamWriter (abstract)

PipedWriter

PrintWriter

StringWriter

FileWriter

Figure 4 – Writers

* Please have a look over: <https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html>

Reader (abstract)

BufferedReader

CharArrayReader

FilterReader (abstract)

InputStreamReader (abstract)

PipedReader

StringReader

FileReader

LineNumberReader

PushBackReader

Figure 5 – Readers

## RandomAccessFile

* This class offers the possibility to read and write in the same time. This is not possible using FileInputStream or FileOutputStream;
* The following code snapshots shows some examples for using RandomAccessFile class;
  + Navigate into file;

RandomAccessFile file = **new** RandomAccessFile(**"c:\\data\\file.txt"**, **"rw"**);  
file.seek(200);  
**long** pointer = file.getFilePointer();  
file.close();

* + Reading from file;

RandomAccessFile file = **new** RandomAccessFile(**"c:\\data\\file.txt"**, **"rw"**);  
**int** aByte = file.read();  
file.close();

* + Writing into file;

RandomAccessFile raf = **new** RandomAccessFile(filePath, **"rw"**);  
  
raf.writeUTF(**"primul string;"**);  
raf.writeUTF(**"second string;"**);  
raf.writeUTF(**"third string;"**);  
  
raf.close();  
  
RandomAccessFile file = **new** RandomAccessFile(**"c:\\data\\file.txt"**, **"rw"**);  
file.write(**"Hello World"**.getBytes());  
file.close();

* For more examples please access: <https://docs.oracle.com/javase/8/docs/api/java/io/RandomAccessFile.html>

# Tasks

**Task 1:**

1. Create a singleton class that implements read-write methods into files like the following:

* User – name, dateOfBirth, hashPassword
* Item – name, description, price
* Invoice – code, date, listOfItems
* Make the objects Serializable.

(Use FileOutputStream, FileInputStream, ObjectOutputStream and ObjectInputStream).

1. In the Main class instantiate an object User, two-three objects Item and an Invoice object, save them in different files and then upload them. After that, print the objects in the console.

**Task 2:**

1. Create a singleton class that implements read-write methods into file using RandomAccessFile. (use methods like getString, getInt, writeString, writeInt).
2. Add a method that read from txt file like this:
   1. Read the last row as an int, the obtained number is the number of rows in the file minus the last row.

To read the last row do not read the file line by line.

* 1. Then read all the rows from the beginning to the last and print them in the console.
  2. What errors did you have to catch in the written code?

**Task 3:**

Use Properties class (<https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html> ) and upload a file of properties.

Properties properties = **new** Properties();  
  
**try** {  
 properties.load(**new** FileReader(**new** File(**"my-prop-file.properties"**)));  
  
 **for**(String key : properties.stringPropertyNames()){  
 System.***out***.println(properties.getProperty((String)key));  
 }  
  
 properties.store(**new** PrintStream(**new** File(**"my-prop-file.properties"**)), **"properties"**);  
  
} **catch** (IOException e) {  
 e.printStackTrace();  
}

**Task 4:**

1. Create a singleton class that implements read-write methods into file using Reader and Writer (FileWriter or PrintWriter and FileReader).
2. The methods will throw new exceptions, implement of you.

Ex: writeTextToFile(String fileNamePath, String text) throws WriteLaboratorException

readTextFromFile(String fileNamePath, String text) throws ReadLaboratorException

1. The singleton class will be created in a package named tasks.utils and the exceptions will be into tasks.exceptions.
2. In the Main class call the methods to write a text and read it.