Advanced Programming Methods
Lecture 8

Content

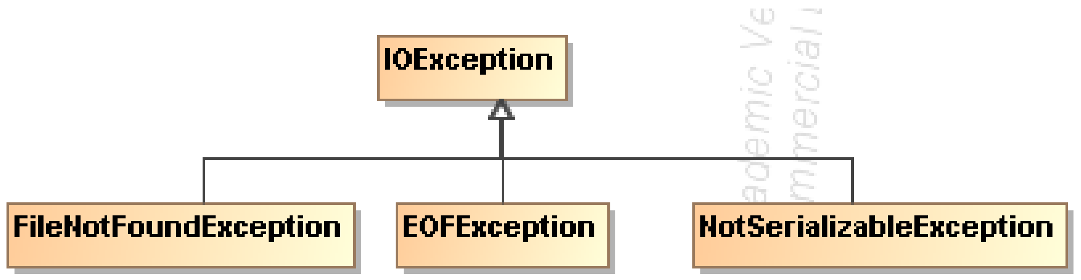
I/O operations in

- Java
- C#

Java I/O

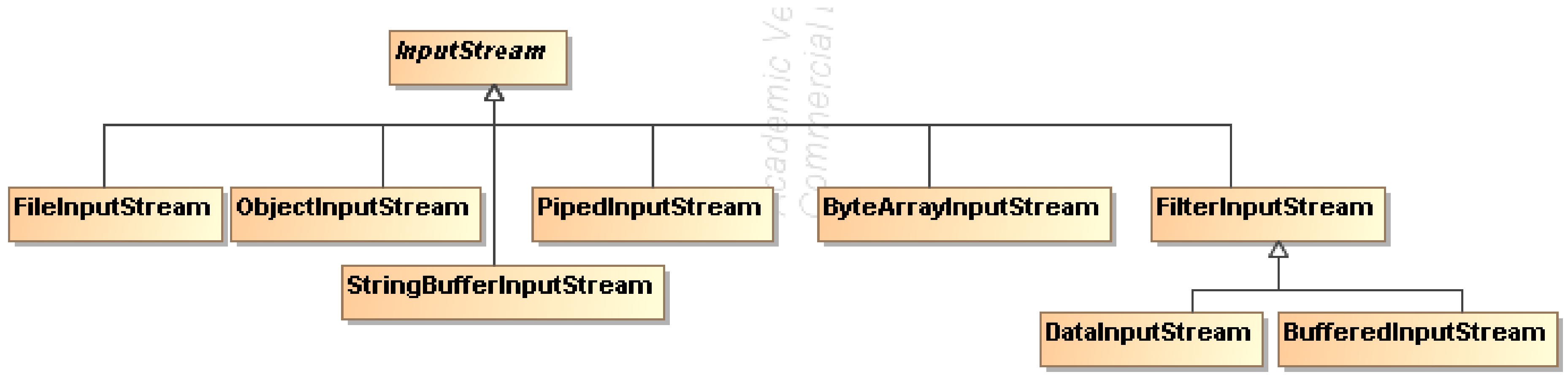
I/O

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



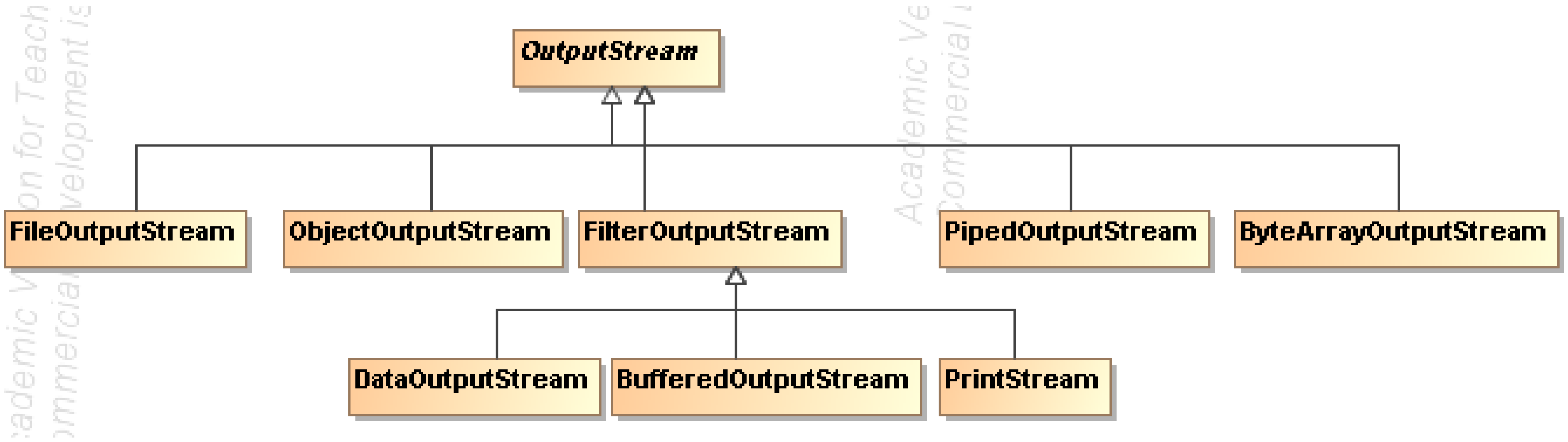
InputStream

- 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

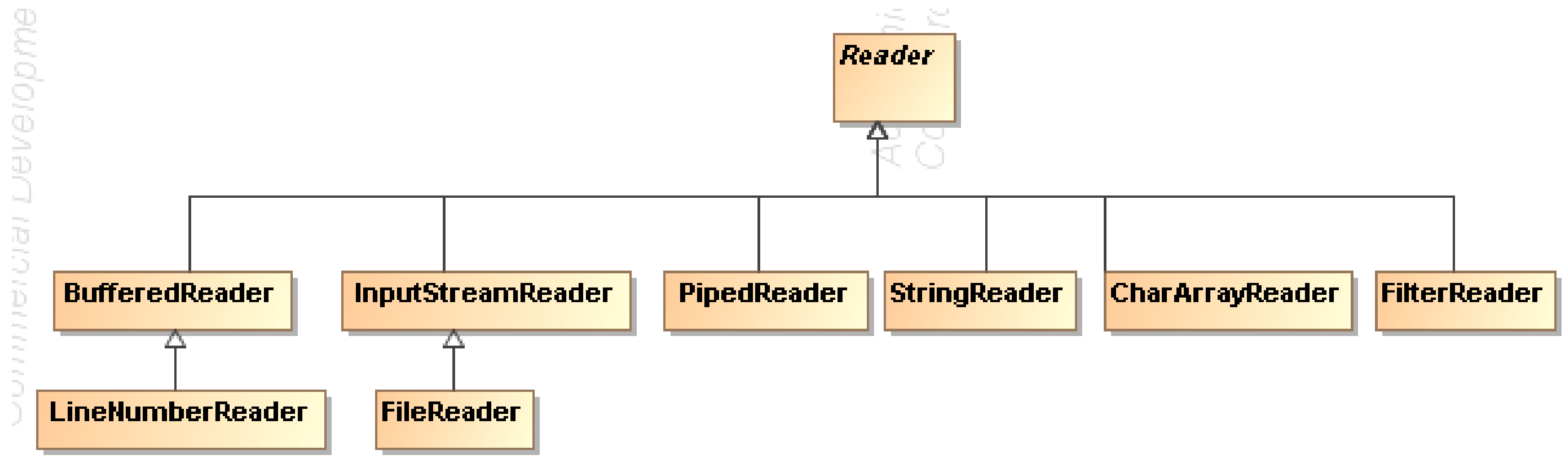


Example

```
 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);}
}
```

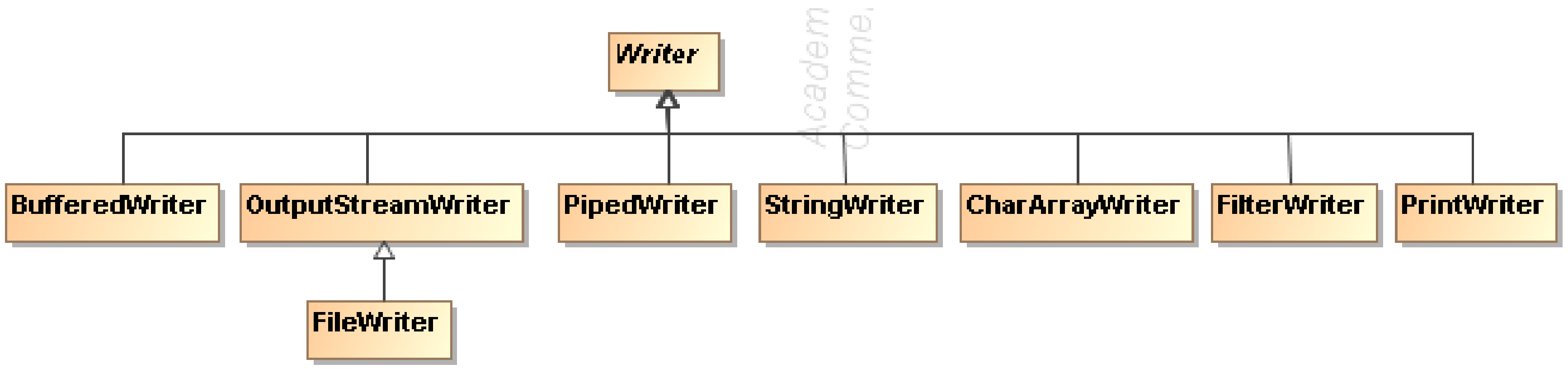
Reader

- 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);}
}
```

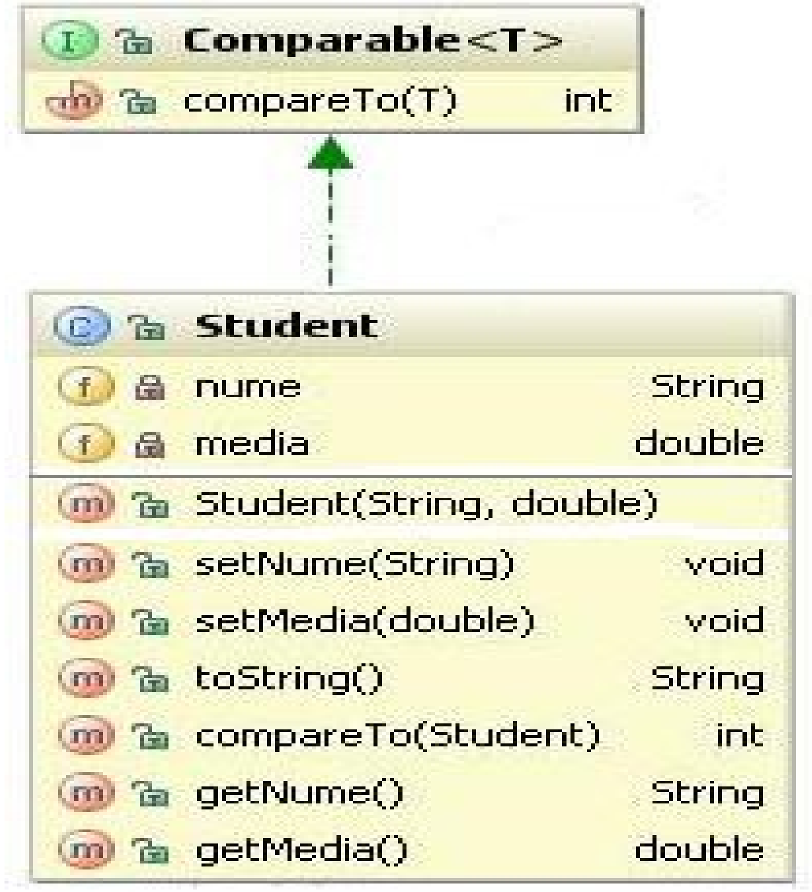
Classes

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

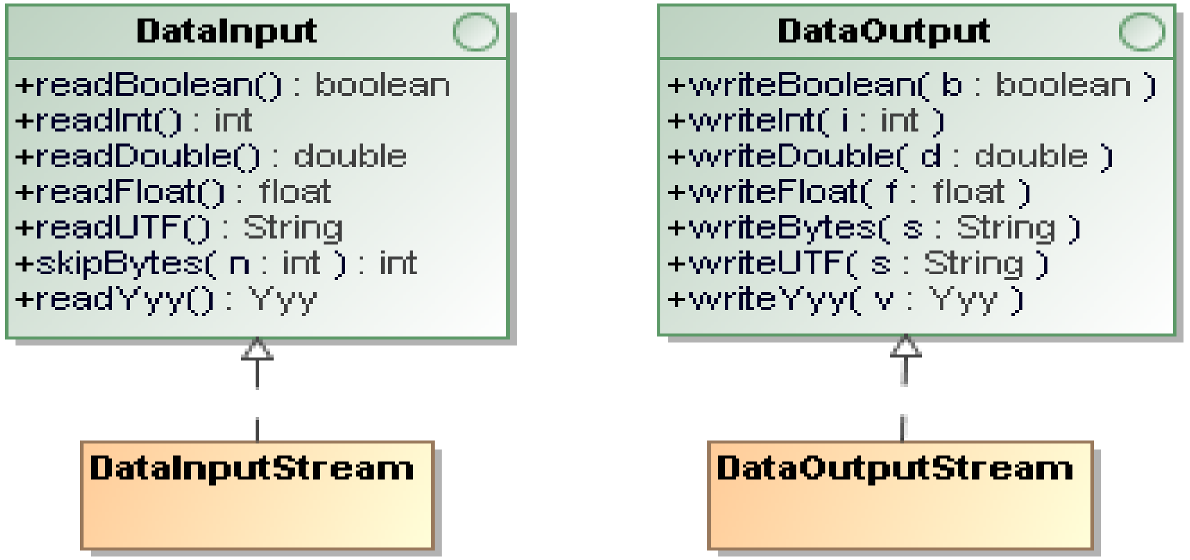
- Read a list of students (from a file, from keyboard)
- Saving the list of students in ascending order based on their average (into a file)

```
//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`

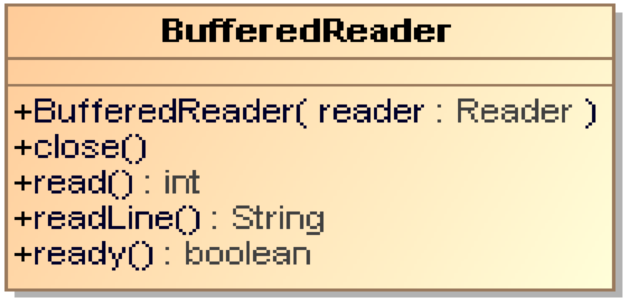
The associated streams can be modified using the methods:

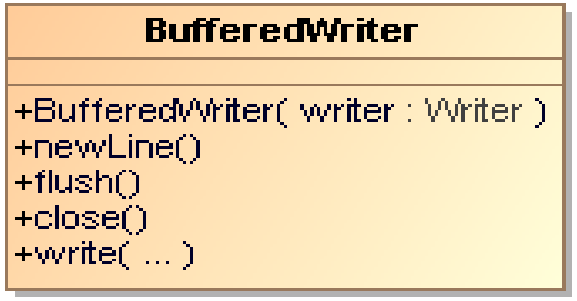
`System.setIn(), System.setOut(), System.setErr(),`

```
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

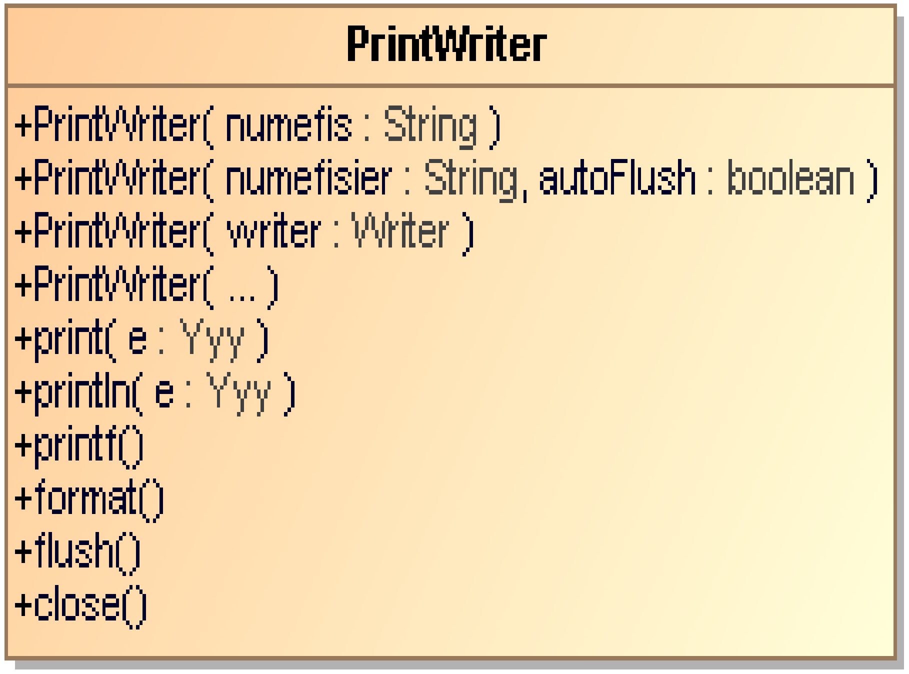
```
List<Student> citesteStudenti(String numefis){
   List<Student> ls=new ArrayList<Student>();
   BufferedReader br=null;
   try{
       br=new BufferedReader(new FileReader(numefis));
       String linie;
       while((linie=br.readLine())!=null){
           String[] elems=linie.split("[|]");
           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 .


PrintWriter
+PrintWriter(numefis : String)
+PrintWriter(numefisier : String, autoFlush : boolean)
+PrintWriter(writer : Writer)
+PrintWriter(...)
+print(e : Yyy)
+println(e : Yyy)
+printf()
+format()
+flush()
+close()

- `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.

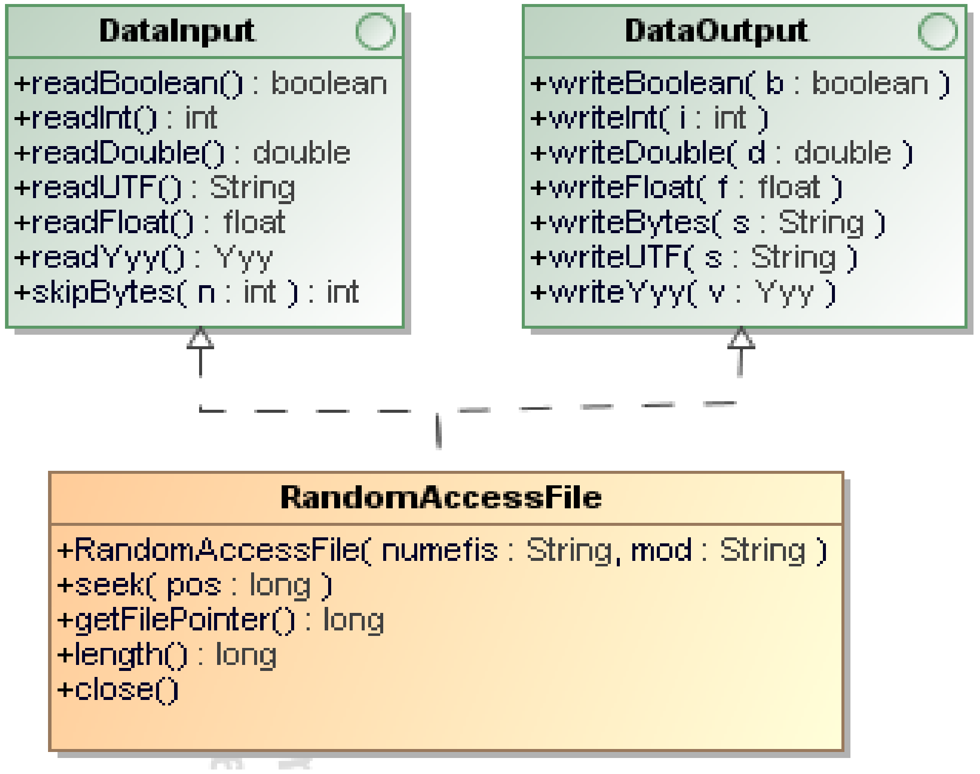
```
List<Student> citesteStudentiScanner(){
  //...
  if (scanner.hasNextDouble()){
    //...
  } else{
        String msj=scanner.nextLine();
        if ("gata".equalsIgnoreCase(msj)){
                        gata=true;
                        continue;
         }else
            System.out.println("Trebuie 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

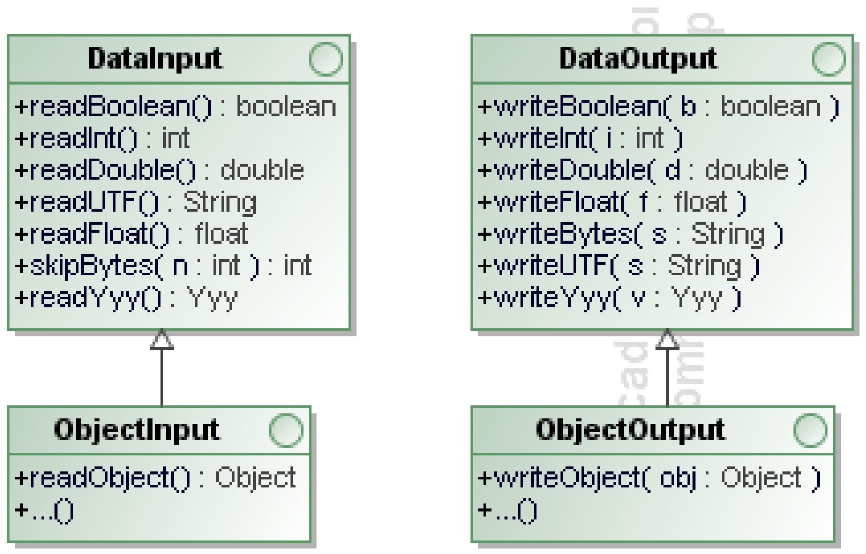
```
List<Student> citesteStudentiRAF(String numefis){
  List<Student> studs=new ArrayList<Student>();
  RandomAccessFile in=null;
  try{
            in=new RandomAccessFile(numefis, "r");
            while(true){
                String nume=in.readUTF();
                double media=in.readDouble();
                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

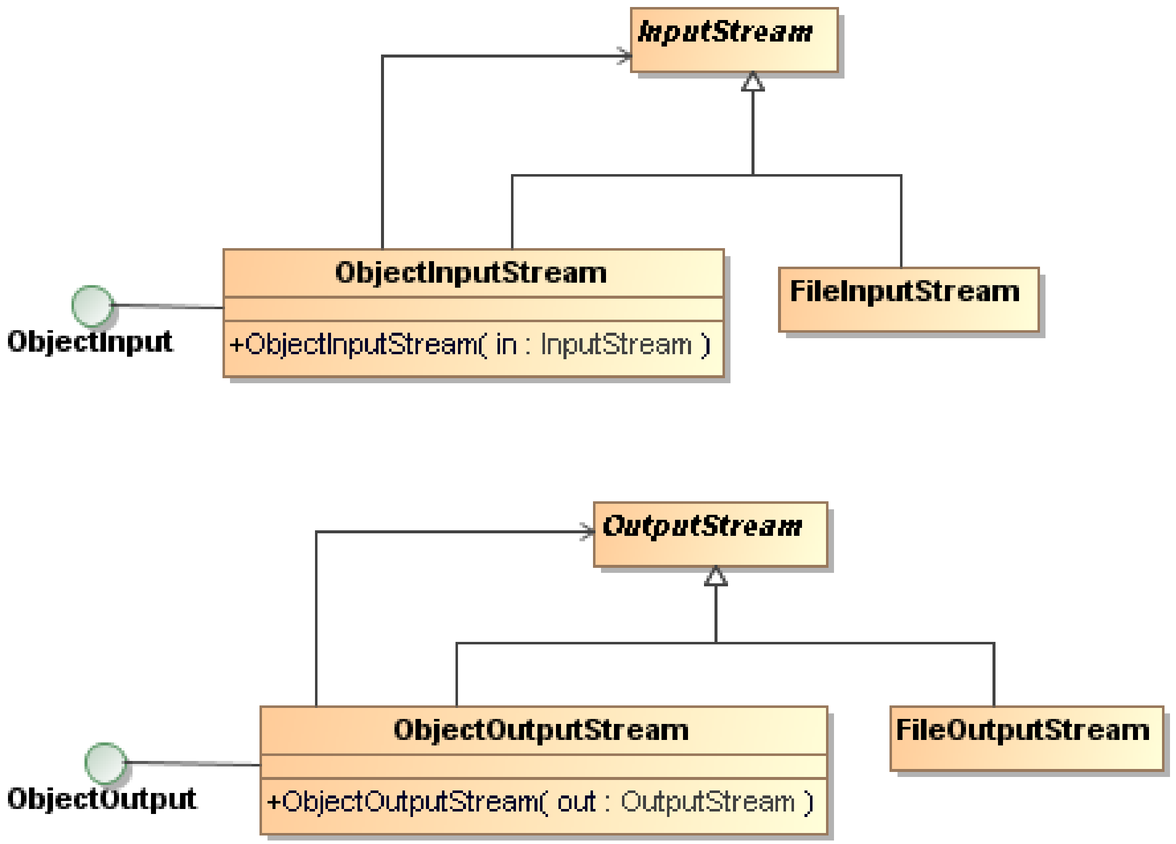
```
void adaugaStudent(Student stud, String numefis){
        RandomAccessFile out=null;
        try{
            out=new RandomAccessFile(numefis, "rw");
            out.seek(out.length());
            out.writeUTF(stud.getNume());
            out.writeDouble(stud.getMedia());
        } catch (FileNotFoundException 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);
                }
        }
    }
```

Objects Serialization

- The process of writing/reading objects from/to a file/external support.
- An object is persistent (serializable) if it can be written into a file/external support and can be read from a file/external support



Objects Serialization



Objects Serialization

```
void serializareObj(String numefis){
        ObjectOutputStream out=null;
        try{
            out=new ObjectOutputStream(new FileOutputStream(numefis));
            out.writeObject(23);
            out.writeObject("Vasilescu Ana");
            out.writeObject(23.45f);
        } catch (IOException e) {
            System.err.println("Eroare "+e);
        } finally {
            if (out!=null)
                try {
                    out.close();
                } catch (IOException e) {
                    System.err.println("Eroare "+e);
                }
        }
    }
```

Objects Serialization

```
void deserializareObj(String numefis){
  ObjectInputStream in=null;
  try{
     in=new ObjectInputStream(new FileInputStream(numefis));
     Integer intreg=(Integer)in.readObject();
     String text=(String)in.readObject();
     Float nr=(Float)in.readObject();
     System.out.println("Intreg: "+intreg+" String: "+text+" Float: "+nr);
  } catch (IOException e) {System.err.println("Eroare "+e);}
   catch (ClassNotFoundException e) {
     System.err.println("Eroare deserializare "+e);
   }finally {
       if (in!=null){
          try {
              in.close();
          } catch (IOException e) {System.err.println("Eroare "+e);}

        }
    }
}
```

Serializable Objects

- The classes whose objects are serializable must be declared to implement the interface `Serializable` (package `java.io`).
- Interface Serializable does not contain any method.

```
class Student implements Comparable<Student>, Serializable{
 //...
}
class Test{
  public static void main(String[] args){
     ObjectOutputStream out=
      //... initialization
      Student stud=new Student(“Popescu Ioan”, 7.9);
      out.writeObject(stud);
      //...
  }
}
```

- The state of `stud` (the values of its fields) is saved into the file.

Serializable objects

- All the reachable objects (the objects that can be reach using the references) are saved into the file only once.

```
class ListaCirculara implements Serializable{
  private class Nod implements Serializable{
      Nod urm;
      //...
  }
   private Nod cap; //last node of the list refers to the head of the list
   //...
}
```

- The objects which are referred by a serializable object must be also serializable.

Obs:

Static attributes of a serializable class are not saved into the file/external support.

Example serializable objects

```
void printSerializabil(List<Student> studs, String numefis){
        ObjectOutputStream out=null;
        try{
          out=new ObjectOutputStream(new FileOutputStream(numefis));
            out.writeObject(studs);
        } catch (IOException e) {
            System.err.println("Eroare serializare "+e );
        } finally {
            if (out!=null)
                try {
                    out.close();
                } catch (IOException e) {
                    System.err.println("Eroare "+e);
                }
        }
    }
```

Example serializable objects

```
@SuppressWarnings("unchecked")
List<Student> citesteSerializabil(String numefis){
        List<Student> rez=null;
        ObjectInputStream in=null;
        try{
            in=new ObjectInputStream(new FileInputStream(numefis));
            rez=(List<Student>)in.readObject();
        } catch (IOException e) {
            System.err.println("Eroare deserializare"+e);
        } catch (ClassNotFoundException e) {
            System.err.println("Eroare deserializare "+e);
        }finally{
          if (in!=null)
                try {
             in.close();
                }catch (IOException e) {System.err.println("Eroare "+e); }
       }
        return rez;
}
```

Objects Serialization

- Method `in.readObject():Object`
  1. Read the object from the stream
  2. Identify the object type
  3. Initialize the non-static members of the object byte by byte (without a constructor call) and then return the new created object
- Method `out.writeObject(Object)`
  - Save the non-static members and the information required by JVM to rebuild the object
  - an object (from a given reference) is saved only once on a stream:

```
ObjectOutputStream out=...
out.writeObject(new Produs("A"));
Produs produs2=new Produs("B");
out.writeObject(produs2);
produs2.setNume(“BB”);
out.writeObject(produs2);
//...
out.close();
```

```
ObjectInputStream in=...
Produs p1=(Produs)in.readObject();
Produs p2=(Produs)in.readObject();
Produs p3=(Produs)in.readObject();
//...
```

Objects Serialization - serialVersionUID

```
public class Student implements Serializable{
  private String nume;
  private double media;
  //...
}
```

Scenario:

1. The objects of class Student are serialized.
2. The class Student is changed (add/remove fields/methods).
3. We want to de-serialize the saved objects.

```
public class Student implements Serializable{
  [any modif access] static final long serialVersionUID = 1L;
   private String nume;
  private double media;
   private int grupa;
  //...
}
```

New added fields are initialized with the default values corresponding to their types.

Objects Serialization - transient

- There are situation when we do not want to save the values of some fields (e.g. passwords, file descriptors, etc.)
- Those fields are declared using the keyword `transient:`

```
public class Student implements Serializable{
  private String nume;
  private double media;
  private transient String parola;
  //...
}
```

At reading, the transient fields are initialized with the default values corresponding to their types.

Serializable data structures

```
public class Stiva implements Serializable{
   private class Nod implements Serializable{
   //...
   }
    private Nod varf;
   //...
 }
//...
Stiva s=new Stiva();
 s.pune(“ana”);
 s.pune(new Produs(“Paine”, 2.3));
                  //class Produs must be serializable
//...
 ObjectOuputStream out=...
  out.writeObject(s);
```

Class File

- Represent the name of a file (not its content).
- Allow platform-independent operations on the files (create, delete, rename, etc.) .
  - `File(nume:String) //nume  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("Director:" + dirCurent.getAbsolutePath());
```

- Creating an SO-independent path

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

- Selecting the .txt files from a directory

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

- Removing a file

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

Package java.nio

- `Version 1.4`
- Read/write operations are more efficient since they use the same data structures as OS: channels and buffers
- Classes:
  - `FileChannel` //channel for files
    - Classes `FileInputStream`, `FileOutputStream` si `RandomAccessFile` have methods that return `FileChannel` objects
  - `ByteBuffer` //used to keep data (read, write)
  - `IntBuffer`, `ShortBuffer`, ...//allow to visualize data of primitive types
  - `MappedByteBuffer` //big files manipulation
  - `FileLock` // to synchronize the file access

Obs:

Classes from package java.io have been rewritten based on the package java.nio in order to improve their performance.

Example java.nio

```
final int BSIZE = 1024;
// writing into a file
        FileChannel fc = new FileOutputStream("data.txt").getChannel();
        fc.write(ByteBuffer.wrap("Ana are mere ".getBytes()));
        fc.close();
// Appending to a file
        fc =new RandomAccessFile("data.txt", "rw").getChannel();
        fc.position(fc.size()); // move at the file end
        fc.write(ByteBuffer.wrap("si pere".getBytes()));
        fc.close();
// Reading from a file
        fc = new FileInputStream("data.txt").getChannel();
        ByteBuffer buff = ByteBuffer.allocate(BSIZE);
        fc.read(buff);
        buff.flip();
        while(buff.hasRemaining())
            System.out.print((char)buff.get());
```

Example java.nio

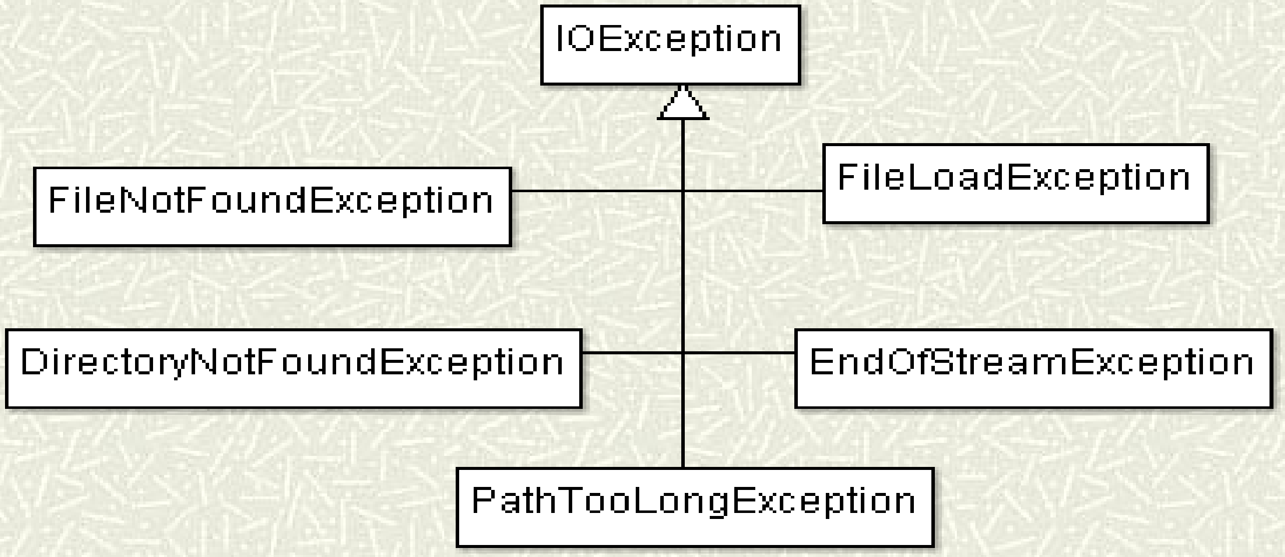
```
//Copy the content of a file into another file
final int BSIZE = 1024;
  FileChannel in = new FileInputStream(args[0]).getChannel(),
  FileChannel out = new FileOutputStream(args[1]).getChannel();
  ByteBuffer buffer = ByteBuffer.allocate(BSIZE);
  while(in.read(buffer) != -1) { //read BSIZE bytes
    buffer.flip(); // prepare the buffer for writing
    out.write(buffer);
    buffer.clear(); // prepare the buffer for reading
 }
```

C# I/O

I/O

Namespace `System.IO`

Exceptions



Utility I/O Classes

The `System.IO` namespace provides a set of types for performing utility file and directory operations, such as copying, moving, creating directories, and setting file attributes and permissions.

Static classes: `File` and `Directory`

Instance method classes (constructed with a file or directory name): `FileInfo` and `DirectoryInfo`

Static class, `Path`, that provides string manipulation methods for filenames and directory paths.

The static methods on `File` and `Directory` are convenient for executing a single file or directory operation.

File / FileInfo

```
public static class File{
    bool Exists (string path);
    void Delete  (string path);
    void Copy    (string sourceFileName, string destFileName);
    void Move    (string sourceFileName, string destFileName);
    void Replace (string source, string destination, String backup);
    FileAttributes GetAttributes (string path);
    void SetAttributes(string path, FileAttributes fileAttributes);
    DateTime GetCreationTime   (string path);
    FileSecurity GetAccessControl (string path);
    void SetAccessControl (string path, FileSecurity fileSecurity);
    //…
}
```

- `FileInfo` offers most of the `File's` static methods in instance form—with some additional properties such as `Extension`, `Length`, `IsReadOnly`, and `Directory` that returns the corresponding `DirectoryInfo` object.

Directory / DirectoryInfo

```
public static class Directory{
    static static bool Exists(string path);
    static void Move(string sourceDirName,string destDirName)
    static string GetCurrentDirectory ();
    static void   SetCurrentDirectory (string path);
    static DirectoryInfo CreateDirectory  (string path);
    static DirectoryInfo GetParent        (string path);
    static string[] GetLogicalDrives();
    static string[] GetFiles(string path);
    static string[] GetDirectories(string path);
    static void Delete(string path)
    //…
}
```

- `DirectoryInfo` exposes instance methods for creating, moving, and enumerating through directories and subdirectories.

Stream Architecture

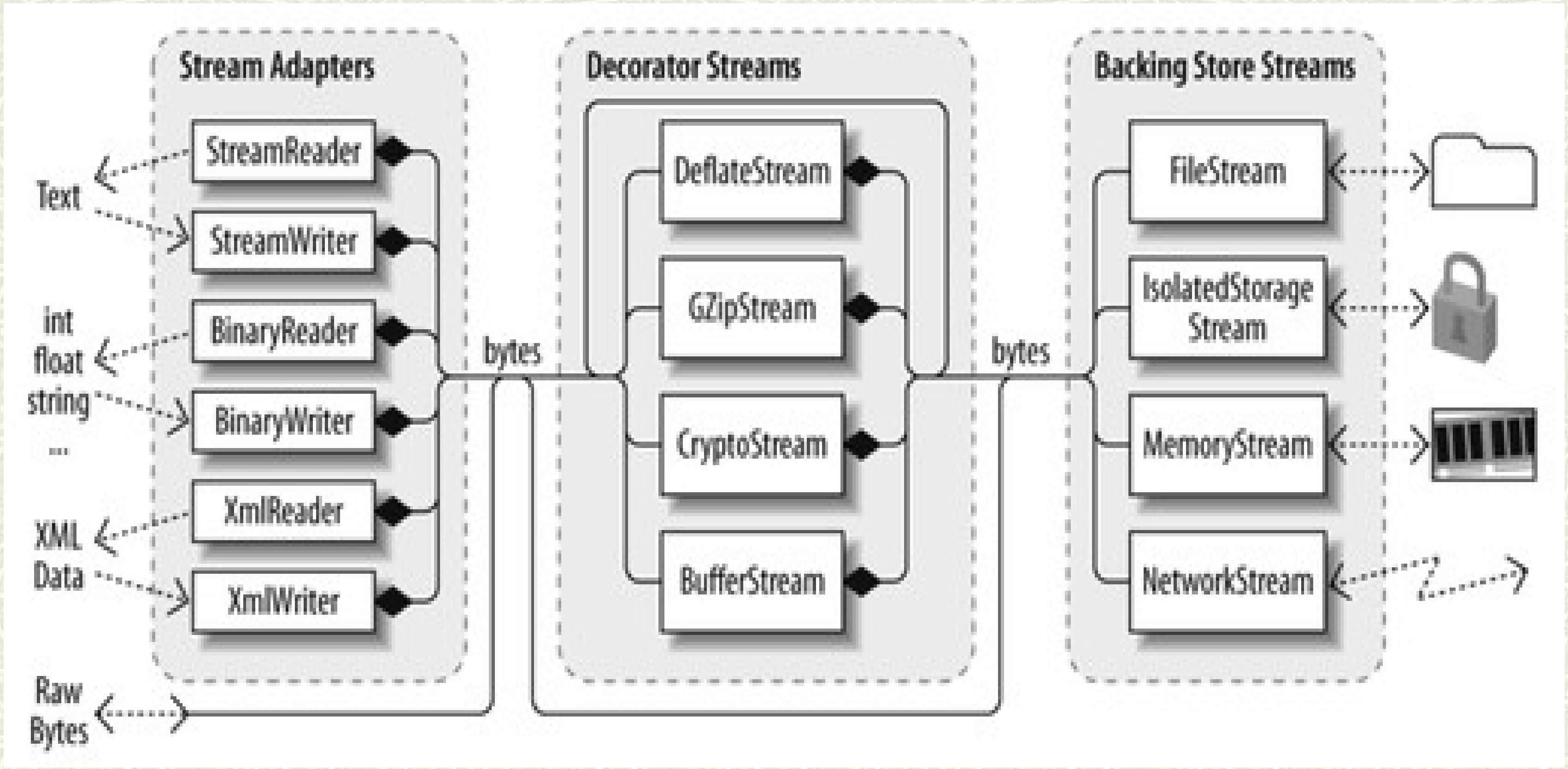
The .NET stream architecture centers on three concepts: backing stores, decorators, and adapters:

- A *backing store* is the endpoint that makes input and output useful, such as a file or network connection. It is one or both of the following:
  - A *source* from which bytes can be sequentially read.
  - A *destination* to which bytes can be sequentially written.
- *Decorator streams* feed off another stream, transforming the data in some way.
- *Adapter* wraps a stream in a class with specialized methods typed to a particular format.

Remark:

An adapter wraps a stream, just as a decorator. Unlike a decorator, however, an adapter is not itself a stream; it typically hides the byte-oriented methods completely.

Stream Architecture



`Stream` class

The abstract `Stream` class is the base for all streams. It defines methods and properties for three fundamental operations: reading, writing, and seeking, as well as for administrative tasks such as closing, flushing, and configuring timeouts.

Stream class

Reading	`public abstract bool CanRead { get; }`
	`public abstract int Read (byte[] buffer, int offset, int count)`
	`public virtual int ReadByte( );`

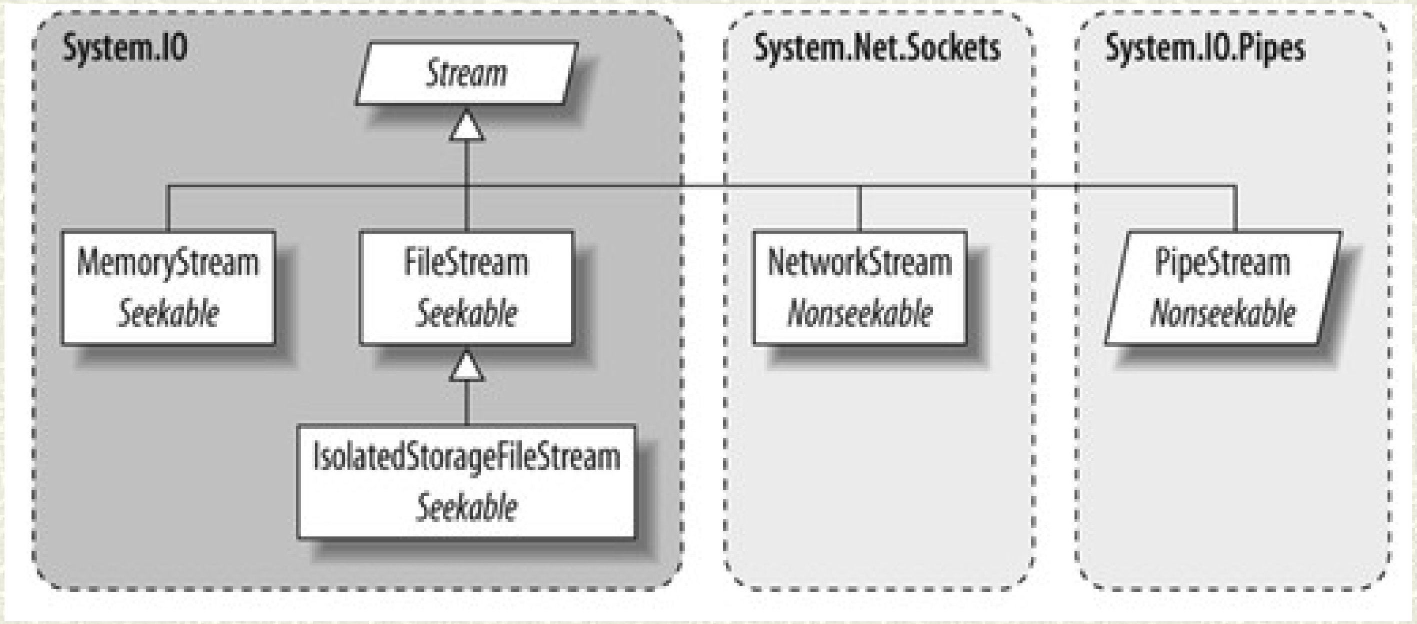
Writing	`public abstract bool CanWrite { get; }`
	`public abstract void Write (byte[] buffer, int offset, int count);`
	`public virtual void WriteByte (byte value);`

Seeking	`public abstract bool CanSeek { get; }`
	`public abstract long Position { get; set; }`
	`public abstract void SetLength (long value);`
	`public abstract long Length { get; }`
	`public abstract long Seek (long offset, SeekOrigin origin);`

Closing/ flushing	`public virtual void Close( );`
	`public void Dispose( );`
	`public abstract void Flush( );`

Timeouts	`public abstract bool CanTimeout { get; }`
	`public override int ReadTimeout { get; set; }`
	`public override int WriteTimeout { get; set; }`

Backing Store Streams



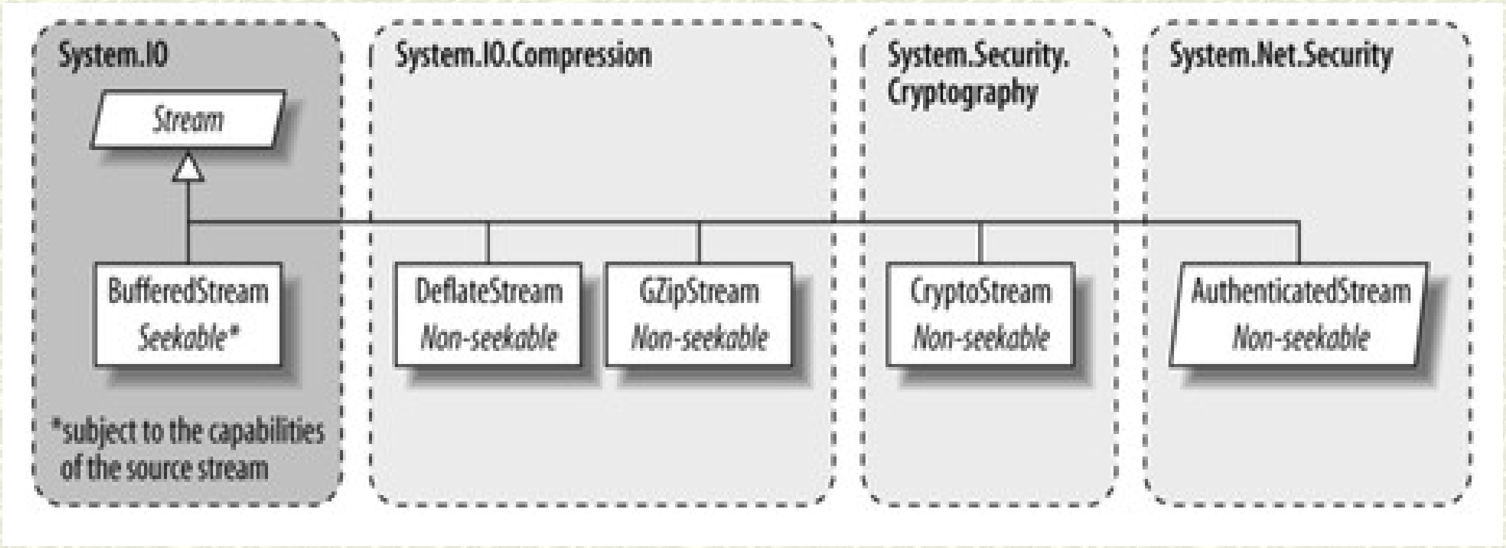
FileStream class

```
public class FileStream : Stream{
    public FileStream(string path, FileMode mode); //overloaded
    public override int Read (byte[] buffer, int offset, int count);
    public override int ReadByte( );
    public override long Seek (long offset, SeekOrigin origin);
    //…
}
public enum SeekOrigin{Begin, Current, End}

public enum FileMode{CreateNew, Create, Open, OpenOrCreate,  Truncate,  Append
}
```

FileStream class - example

```
using(FileStream fs=new FileStream("testFileStream.txt", FileMode.Create)) {
  String message = "Ana are mere.";
    //helper class to transform a string in an array of bytes, System.Text
  UnicodeEncoding unienc=new UnicodeEncoding();
  //writing to the file
  fs.Write(unienc.GetBytes(message), 0, unienc.GetByteCount(message));

     //creates an array of bytes for reading the data from the file
   byte[] rbytes=new byte[fs.Length];
     //position the file pointer to the beginning
   fs.Seek(0, SeekOrigin.Begin);
     //reads the data from the file
   fs.Read(rbytes, 0, (int) fs.Length);
     //transforms the bytes in a string
   String newMess=new string(unienc.GetChars(rbytes,0,rbytes.Length));
   Console.WriteLine("Written text {0}, read text {1}", message, newMess);
}
```

Decorator Streams



Stream Adapters

A `Stream` deals only with bytes.

In order to read or write data types such as `strings`, `integers`, or XML elements a stream adapter should be used:

Text adapters (for string and character data):

- `TextReader`, `TextWriter`,
- `StreamReader`, `StreamWriter`,
- `StringReader`, `StringWriter`

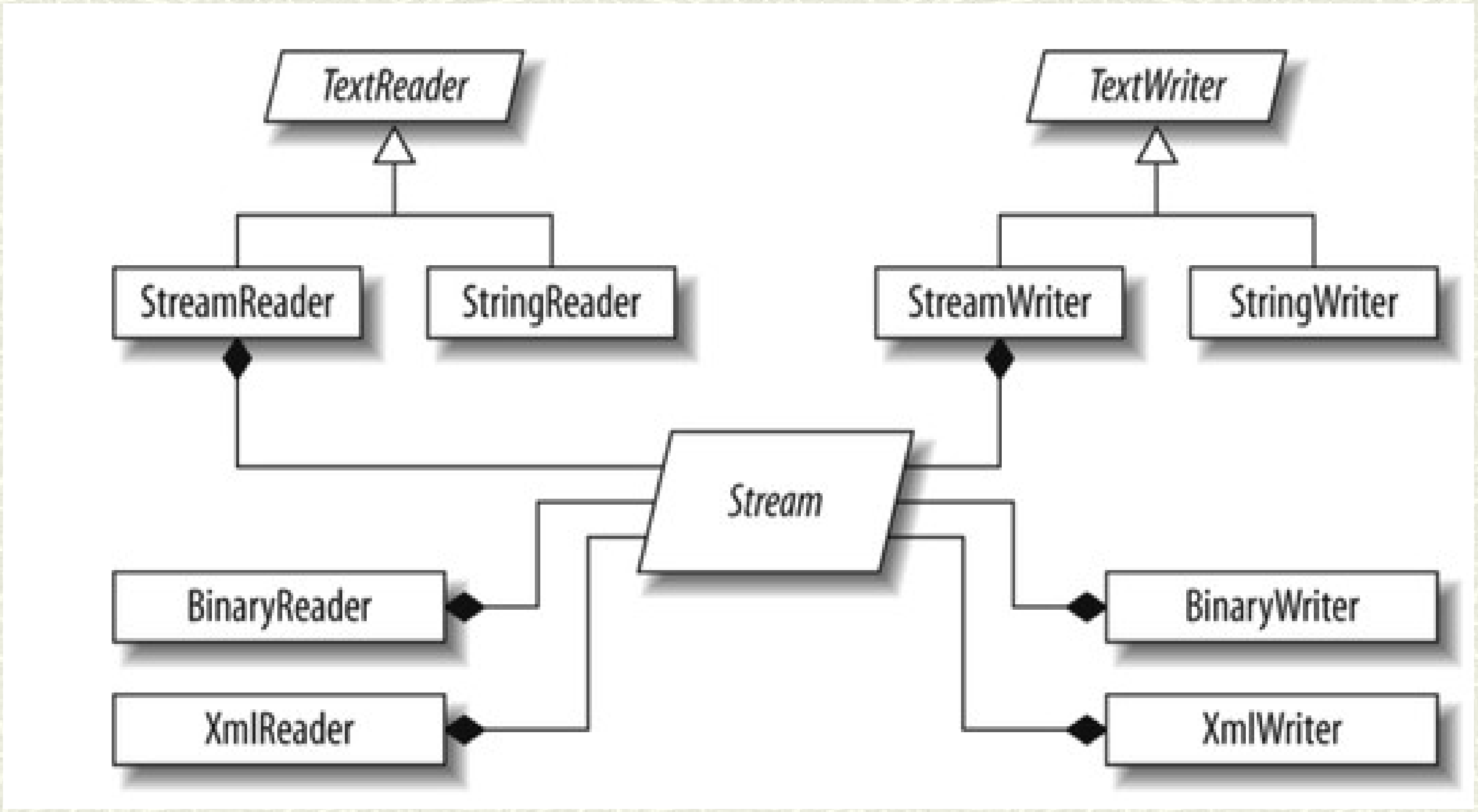
Binary adapters (for primitive types such as `int`, `bool`, `string`, and `float`)

- `BinaryReader`, `BinaryWriter`

XML adapters

- `XmlReader`, `XmlWriter`

Stream Adapters



Text Adapters

- `TextReader` and `TextWriter` are the abstract base classes for adapters that deal exclusively with characters and strings. Each has two general-purpose implementations in the framework:

`StreamReader`/`StreamWriter`: uses a `Stream` for its data store, and translates the stream's bytes into characters or strings

`StringReader`/`StringWriter`: implements `TextReader`/`TextWriter` using in-memory strings

Because text adapters are often associated with files, the `File` class provides the static methods `CreateText`, `AppendText`, and `OpenText` to ease the process of creating file-based text adapters.

Text Adapters

```
//Writing to a text file
using (FileStream fs = File.Create ("test.txt"))
using (TextWriter writer = new StreamWriter (fs))
//or TextWriter writer=File.CreateText(“test.txt”)
{
  writer.WriteLine (“Ana are mere.");
  writer.WriteLine (“mere");
}
//Reading from a text file
using (FileStream fs = File.OpenRead ("test.txt"))
using (TextReader reader = new StreamReader (fs))
//or TextReader reader=File.OpenText(“test.txt”)
{
  Console.WriteLine (reader.ReadLine());       // reads line 1
  Console.WriteLine (reader.ReadLine());       // reads line 2
}
```

Binary Adapters

- **`BinaryReader`** and **`BinaryWriter`** read and write native data types: **`bool`**, **`byte`**, **`char`**, **`decimal`**, **`float`**, **`double`**, **`short`**, **`int`**, **`long`**, **`sbyte`**, **`ushort`**, **`uint`**, and **`ulong`**, as well as **`strings`** and **`arrays`** of the primitive data types.

```
using (BinaryWriter w = new BinaryWriter(File.Create("testBinary.txt")))
{
       w.Write("Ana");
       w.Write(23);
       w.Write(12.4);
       w.Flush();
}
using(BinaryReader r = new BinaryReader(File.Open("testBinary.txt",FileMode.Open)))
 {
       String name = r.ReadString();
       int age = r.ReadInt32();
       double d= r.ReadDouble();
       Console.WriteLine("name= {0}, age={1}, d={2}",name,age,d);
  }
```

Serialization

Serialization is the act of taking an in-memory object or object graph (set of objects that reference each other) and flattening it into a stream of bytes or XML nodes that can be stored or transmitted. *Deserialization* takes a data stream and reconstitutes it into an in-memory object or object graph.

Serialization and deserialization are typically used to:

Transmit objects across a network or application boundary.

Store representations of objects within a file or database.

There are four serialization mechanisms in the .NET Framework:

The data contract serializer (.NET 3.0, Windows Communication Foundation)

The binary serializer

The (attribute-based) XML serializer (`XmlSerializer`)

The `IXmlSerializable` interface

The Binary Serializer

The binary serialization engine is used implicitly by Remoting.

The binary serialization is highly automated and can handle complex object graphs with minimum intervention.

There are two ways to make a type support binary serialization:

Attribute-based, using `[Serializable]`

`ISerializable` interface.

A type can be made serializable with a single attribute:

```
[Serializable] public class Person
{
   string Name;
    int Age;
 public Person(String n,int a){…}
 //…
}
```

Serializable Attribute

Remarks:

The `[Serializable]` attribute instructs the serializer to include all fields in the type. This includes both private and public fields (but not properties).

Every field must itself be serializable; otherwise, an exception is thrown. Primitive .NET types such as `string` and `int` support serialization (as do many other .NET types).

The `Serializable` attribute is not inherited, so subclasses are not automatically serializable, unless also marked with this attribute.

```
[Serializable] class Person{
 //…
}
[Serializable] class Student : Person{
 //…
}
```

Serialization Formatters

In order to serialize an instance of a particular type, a formatter must be instantiated first, and then call the `Serialize` method.

There are two formatters that can be used with the binary engine:

`BinaryFormatter`: it is the more efficient of the two, producing smaller output in less time. Its namespace is `System.Runtime.Serialization.Formatters.Binary`.

`SoapFormatter`: it supports basic SOAP-style messaging when used with Remoting. Its namespace is `System.Runtime.Serialization.Formatters.Soap`.

- `BinaryFormatter` is contained in `mscorlib`;
- `SoapFormatter` is contained in `System.Runtime.Serialization.Formatters.Soap.dll`. The user must add a reference to this assembly when using `SoapFormatter`.

Serialization - Example

//Serializes a Person with a BinaryFormatter:

```
Person p = new Person("George", 25 );
IFormatter formatter = new BinaryFormatter(  );
using (FileStream s = File.Create ("serialized.bin"))
  formatter.Serialize (s, p);
```

//Deserializes(Restores) the object of type Person

```
using (FileStream s = File.OpenRead ("serialized.bin"))
{
  Person p2 = (Person) formatter.Deserialize (s);
  Console.WriteLine (p2.Name + " " + p2.Age);     // George 25
}
```

Binary Serialization Attributes

The `[NonSerialized]` attribute marks a field as non serializable.

Nonserialized members are always empty or null when deserialized, even if field initializers or constructors set them otherwise.

```
[Serializable] public class Person{
  public string Name;
  public DateTime DateOfBirth;
   [NonSerialized] public int Age;
}
```

Deserialization bypasses all constructors and field initializers. `[OnDeserializing]` and `[OnDeserialized]` attributes can be used to initialize fields. They are added in front of the methods to be called.

- `[OnSerializing]` and `[OnSerialized]` flag a method for execution before or after serialization.

Binary Serialization Attributes

- [OptionalField]. By default, adding a field breaks compatibility with data that was already serialized, unless the [OptionalField] attribute is attached to the new field.

```
[Serializable] public class Person {      // Version 1
   string name;
}
[Serializable] public class Person{       // Version 2
   string name;
   int age;
}
[Serializable] public class Person{       // Version 2 Robust
  string name;
  [OptionalField (VersionAdded = 2)] int age; //
}
```

The deserializer treats the `age` field as nonserialized. `Age` is initialized with the default value.

ISerializable Interface

- Implementing `ISerializable` gives a type complete control over its binary serialization and deserialization.

```
public interface ISerializable{
  void GetObjectData (SerializationInfo info, StreamingContext context);
}
```

- `GetObjectData` is called when performing serialization. The code should populate the `SerializationInfo` object (a name-value dictionary) with data from all fields that must serialized.
- In addition to implementing `ISerializable`, a type controlling its own serialization needs to provide a deserialization constructor that takes the same two parameters as `GetObjectData`. The constructor can be declared with any accessibility.
- The implementing class must still use the `[Serializable]` attribute.

ISerializable - example

```
[Serializable] class Person:ISerializable {
        private string name;
        private int age;
        public Person(String name,int age) {
            this.name = name;
            this.age = age;
        }
        private Person(SerializationInfo info, StreamingContext context)      {
            name = info.GetString("name");
            age = info.GetInt32("age");
        }
 public virtual void GetObjectData(SerializationInfo info, StreamingContext
    context) {
    info.AddValue("name", name);
         info.AddValue("age", age);

        }
// …
}
```