Programmierpraktikum

Claudius Gros, SS2012

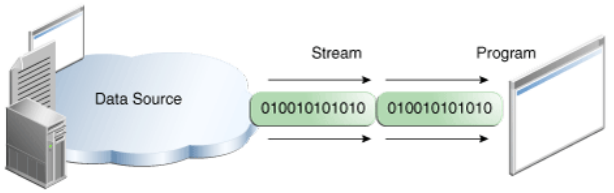
Institut für theoretische Physik
Goethe-University Frankfurt a.M.

Input / Output Streams

Java I/O streams

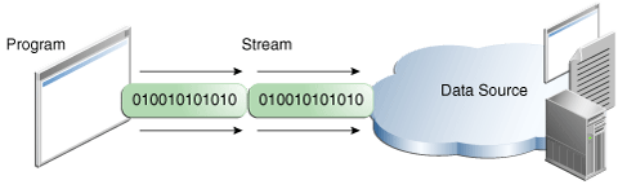
stream from/to everywhere

- screen
- file
- internet connection
- ...



transmit all kind of data

- byte, char, int, ...
- string, objects, ...



java.io

IO - communiation with the world

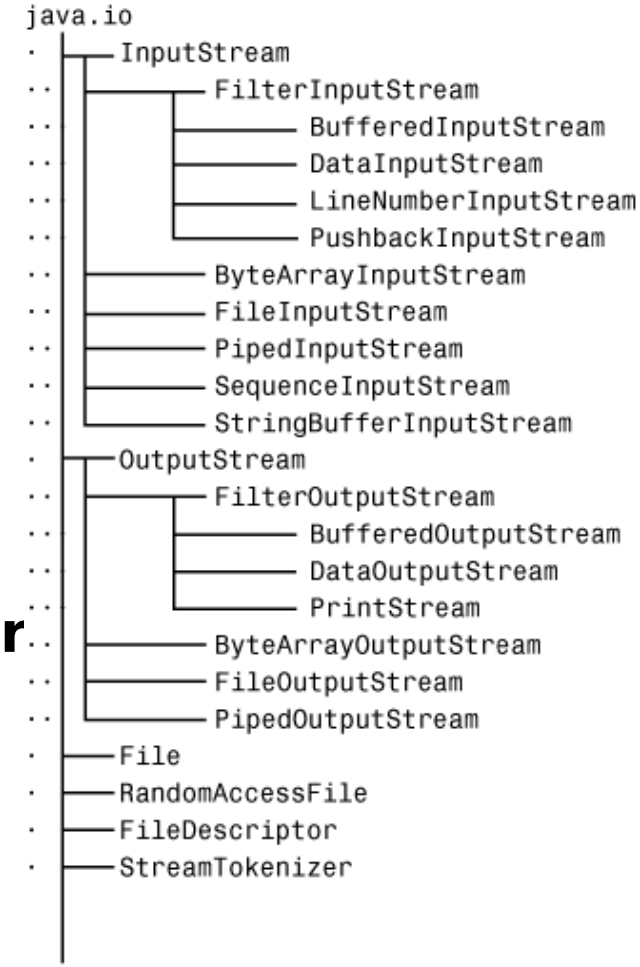
- input / output
- a class for every type of stream

modern programming

- know what exists
- look it up

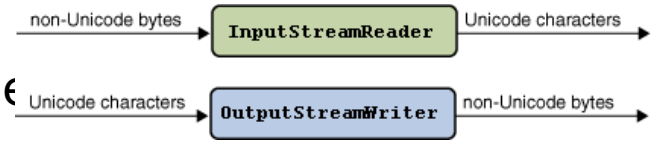
exception handeling mandatory for

- input files may not be present
- possibly no writing permissions on output
- ...



byte streams

IO byte per byte



- input/output byte by byte is low level, inefficie
- **`read()`** returns integer (Unicode, table)
  `: -1` at end
- **`null`**: object not yet been instantiated

```
 1. import java.io.FileInputStream;
 2. import java.io.FileOutputStream;
 3. import java.io.IOException;
 4. //  normally just use   import java.io.*;
 5.
 6. public class CopyBytes {
 7.   public static void main(String[] args) throws IOException {
 8. // what happens if you would just use
 9. // public static void main(String[] args) {
10.
11.    FileInputStream in = null;
```

```
12.    FileOutputStream out = null;
13.
14.    in = new FileInputStream("input.txt");
15.    out = new FileOutputStream("output.txt");
16.    int c;
17.
18. // --- start loop over all bytes in input file
19.     while ((c = in.read()) != -1)
20.       out.write(c);
21.
22. // --- all reading has been done - close streams
23.     if (in != null)    in.close();
24.     if (out != null)  out.close();
25.   }  // end of CopyBytes.main()
26. }  // end of CopyBytes
```

- appending to output file with **`true`**
- **`catch`** exception

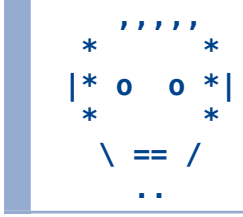
```
1. import java.io.*;
2.
```

```
 3. public class AppendBytes {
 4. public static void main(String[] args)  // no exception thrown
 5.   {
 6.   try {
 7.       FileInputStream inStream = new FileInputStream("input.txt");
 8.       FileOutputStream outStream = new FileOutputStream("output.txt",true);  // append
 9. //               are equivalent:   new FileOutputStream("output.txt",false);
10. //                                 new FileOutputStream("output.txt");
11. //                                 new FileOutputStream("output.txt",(1==2));
12.       int c;
13.       while ((c = inStream.read()) != -1)
14.         outStream.write(c);
15.       } catch(IOException e)
16.         {
17.         System.out.println(e.toString());
18.         }
19.   }  // end of AppendBytes.main()
20. }  // end of AppendBytes
```

- here a possible *input.txt* file

```
This is an text input file
```

with a picture



formatted output

- **`PrintWriter`**:
  has member function **`.print()`**
  `.printf()`
- formatted output is alway sent to a buffer,
  and not directly to the output file
- use member function **`.flush()`**
  to sent bufffer to file

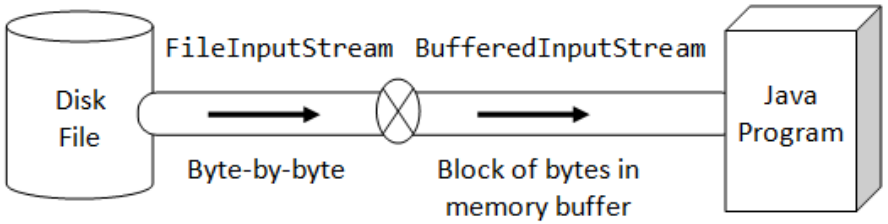
```
1. import java.io.*;
2.
3. public class FormattedOutput {
4. public static void main(String[] args)
5.   {
6.   try {
7.       PrintWriter myOutput = new PrintWriter("output.txt");
8.       for (int i=0;i<8;i++)
9.         myOutput.printf("%2d modulo 3 is %2d\n",i,i%3);
```

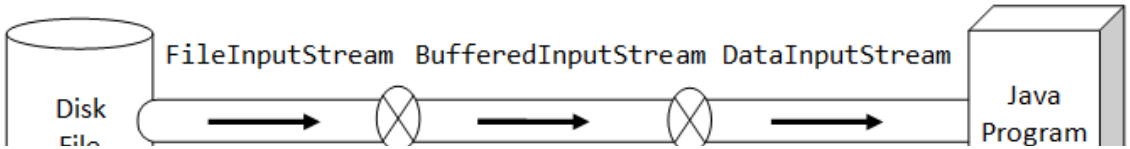
```
10.       myOutput.flush();     // sent buffer to file
11.       } catch(IOException e)
12.         {
13.         System.out.println(e.toString());
14.         }
15.   }  // end of FormattedOutput.main()
16. }    // end of FormattedOutput
```

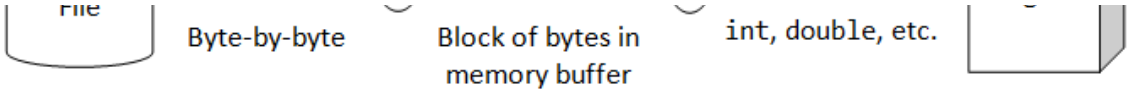
buffered input

buffering

- every read/write request to disk is slow
  (try to read/write a 100 GByte file without buffering)
- **buffering**
  systems reads/writes large chunks of data to file at once
- **`BufferedReader`**:
  wrapper around a **`Reader`**
  has member function **`.readLine()`**






File
Byte-by-byte
Block of bytes in memory buffer
int, double, etc.

buffered input with formatted output

- **BufferedReader(Reader** *in***)**
  readers: **FileInputStream**: byte **Reader**: 8 bits
  FileReader: char **Reader**: 16 bits
- **PrintWriter(File** *file***)** writing via File object
  PrintWriter(OutputStream *out***)** writing via OutputStream
  PrintWriter(String *fileName***)** writing directly to file named *fileName*
  PrintWriter(Writer *out***)** writing via Writer
  PrintWriter(Writer *out***, boolean** *autoFlush***)** writing via Writer with (if *true*) autoflush
- reading **String** data line by line

```
import java.io.FileReader;       // character reader
import java.io.FileWriter;       // character writer
import java.io.BufferedReader;   // buffering input
import java.io.PrintWriter;      // formatting output
```

```
import java.io.IOException;

public class CopyLines {
  public static void main(String[] args) {
    try {
        BufferedReader inputStream = new BufferedReader(new FileReader("input.txt"));
        PrintWriter outputStream = new PrintWriter(new FileWriter("output.txt"));  //
//      PrintWriter outputStream = new PrintWriter("output.txt");

        String s;
        while ((s = inputStream.readLine()) != null)
           outputStream.println(s);
        outputStream.flush();         // send buffer to file

        } catch(IOException e)
        {
        System.out.println(e.toString());
        }
  }   // end of CopyLines.main()
}     // end of CopyLines
```

buffered and formatted ouput

- a must: buffering input/output for large data sizes

```
 1. import java.io.*;
 2.
 3. public class BufferedOutput {
 4. public static void main(String[] args)
 5.   {
 6.   try {
 7.       PrintWriter myOutput = new PrintWriter(                    // for formatting
 8.                                  new BufferedWriter(             // for buffering
 9.                                     new FileWriter("output.txt"))); // ouput stream
10.
11.       for (int i=0;i<8;i++)
12.         myOutput.printf("%2d modulo 5 is %2d\n",i,i%5);
13.       myOutput.flush();     // sent buffer to file
14.
15.       } catch(IOException e)
16.         {
17.         System.out.println(e.toString());
```

```
18.         }
19.   }  // end of BufferedOutput.main()
20. }    // end of BufferedOutput
```

IO streams as arguments in function calls

- printing to any stream -
  avoiding dublicate printing routines
- any object can be an argument - IO streams too
- method needs to throw the corresponding exception

```
 1. import java.io.*;
 2.
 3. public class PrintWriterDemo {
 4.
 5. /** Use a single facility to send data to different streams,
 6.   * note the mandatory exception handling.
 7.   */
 8. static void printToStream(PrintWriter myOutput, String text)
 9.             throws IOException
10.  {
11.  myOutput.printf("this is the text: %s\n",text);
12.  }
```

```
13.
14. /** Always use buffered IO, it never hurts.
15.   */
16.
17. public static void main(String args[])
18.   {
19.   try {
20.     PrintWriter toFile =
21.       new PrintWriter(new BufferedWriter(
22.                       new FileWriter("printWriterDemo.out")));
23.     PrintWriter toConsole =  new PrintWriter(System.out, true);
24.
25.   PrintWriterDemo.printToStream(toFile,"sending data to file");
26.   PrintWriterDemo.printToStream(toConsole,"sending log data to console");
27.
28.   toFile.close();
29.   toConsole.close();
30.       } catch (IOException e) { e.printStackTrace(); }
31.   } // end of PrintWriterDemo.main()
32. } // end of class PrintWriterDemo
```

scientific data files

- always include data description in comment section

  are ignored by most plotting utilities
- typical science data file: *planetsData.list*
- how to read lists of numbers?

```
 1. 9
 2. Mercury 0.0553 4880 5.43 0.000 58.81 d 0.387 87.97 d 0.2056 7.0 0.1
 3. Venus 0.815 12104 5.20 0.000 243.69 d 0.723 224.70 d 0.0068 3.4 177.3
 4. Earth 1.000 12742 5.52 0.0034 23.9345 h 1.000 365.26 d 0.0167 0.00 23.45
 5. Mars 0.107 6780 3.93 0.0065 24.623 h 1.524 686.98 d 0.0934 1.85 25.19
 6. Jupiter 317.83 139822 1.33 0.0649 9.925 h 5.203 11.86 y 0.04845 1.305 3.12
 7. Saturn 95.162 116464 0.687 0.098 10.50 h 9.539 29.46 y 0.05565 2.489 26.73
 8. Uranus 14.536 50724 1.32 0.023 17.24 h 19.182 84.01 y 0.0472 0.773 97.86
 9. Neptune 17.147 49248 1.64 0.017 16.11 h 30.06 164.79 y 0.00858 1.773 29.56
10. Pluto 0.0021 2274 2.05 0.0 6.405 d 39.53 247.68 y 0.2482 17.15 122.46
```

scanning data input files

scanning for data: double, int, ...

- **`Scanner(File`** *`file`***`)`** scanning via File object
  `Scanner(InputStream` *`inStream`***`)`** scanning via InputStream
  ...
- **`new Scanner(System.in)`** for console data input
- has member functions
  `Boolean .hasNext()` : has next entry, seperated by blanks
  `String .next()` : next entry as a **`String`** object
  `int .nextInt()` : next entry as a **`int`** primitive
  `double .nextDouble()` : next entry as a **`double`** primitive
  ...
- note: use **`(string1.equals(string2))`**
  for comparing two strings, **`(string1==string2)`** is not secure

```
1. import java.io.*;
```

```
 2. import java.util.Scanner;
 3.
 4. public class DataInput {
 5. public static void main(String[] args)
 6.   {
 7.   try {
 8.       Scanner myScanner = new Scanner(
 9.                                new FileInputStream("planetsData.list"));
10.
11. // --- skippling initial comments
12.       boolean readingComments = true;
13.       while (readingComments)
14.           readingComments = false;
15.
16. // --- reading data
17.       int nData = myScanner.nextInt();
18.       String name;
19.       double mass;
20.       int diameter;
21.
22.       while (myScanner.hasNext())
23.         {
24.         name     = myScanner.next();
```

```
25.         mass     = myScanner.nextDouble();
26.         diameter = myScanner.nextInt();
27.
28.         for (int i=0;i<10;i++) myScanner.next();  // skipping rest of line
29.         }
30.
31.       } catch(IOException e)
32.         {
33.         System.out.println(e.toString());
34.         }
35.   }  // end of DataInput.main()
36. }    // end of DataInput
```

output of objects

object output streams

- all classes descendents of the **Object** class
  instantiated *Serializable* objects can be sent to output in one piece
- all classes can be made *Serializable*

```
1.      public class MyClass implements Serializable{}
2.
```

- **Array** objects are serializable per definition
- on input one needs to check the type of **Object** read
  using *instanceof*
- data files containing serializabled objects are binary and not human readable
- convenient and secure for a complete system dump

```
1. import java.io.*;
```

```
 2.
 3. public class ObjectsIo {
 4. public static void main(String[] args)
 5.   {
 6. // ***
 7. // *** generate objects and write objects to file
 8. // ***
 9.   String[] stringArrayObject = {"This","is","an","array","of","strings"};
10.   int[] intArrayObject = {4444,333,22,1};
11.   try {
12.       ObjectOutputStream myOutput = new ObjectOutputStream(
13.                                     new FileOutputStream("containsObject.txt"));
14.
15.       myOutput.writeObject(stringArrayObject);
16.       myOutput.writeObject(intArrayObject);
17.
18.       } catch(IOException e)
19.         {
20.         System.out.println(e.toString());
21.         }
22. // ***
23. // *** read objects from file
24. // ***
```

```
25.   try {
26.       ObjectInputStream myInput = new ObjectInputStream(
27.                                   new FileInputStream("containsObject.txt"));
28.
29.       Object obj = null;
30.       while ((obj = myInput.readObject()) != null)
31.         {
32.         if (obj instanceof String[])      // String array ?
33.           {
34.           String[] ss = (String[])obj;    // casting Object to String[]
35.           for (int j=0;j<ss.length;j++)
36.             System.out.printf("%s ",ss[j]);
37.           System.out.printf("\n");
38.           }
39.         if (obj instanceof int[])         // int array ?
40.           {
41.           int[] ii = (int[])obj;          // casting Object to int[]
42.           for (int j=0;j<ii.length;j++)
43.             System.out.printf("%d ",ii[j]);
44.           System.out.printf("\n");
45.           }
46.         }
47.
```

```
48.       } catch (EOFException ex) { // EOF exception
49.            System.out.println("End of file reached.");
50.       } catch(IOException e)
51.         {
52.         System.out.println(e.toString());
53.         } catch (ClassNotFoundException e)
54.           {
55.           System.out.println(e.toString());
56.           }
57.
58.   }  // end of ObjectsIo.main()
59. }    // end of ObjectsIo
```

exercise: IO and OOP

Write a program for object oriented input/output.

- define a serializable **`class`** *`Planet`* storing all planet properties
- read the planet data file and
  create an instantiation of *Planet* for every planet
- write all instantiated *Planet* objects to file
- read the *Planet* objects from file,
  print data to console as a formatted table

directory listings

- **`System.getProperty()`** for various properties of operating system
- **`.isDirectory()`** / **`.isFile()`** for **`File`** objects

```
 1. import java.util.*;
 2. import java.io.*;
 3. import java.math.*;
 4.
 5. public class DirectoryHandling {
 6.
 7. /** Give directory path as argument.
 8.   */
 9. public static void main(String args[]) {
10.
11. // ***
12. // *** open file handler / directory listing
13. // ***
14.   String inputDirPath = null;
15.   if (args.length==1)
```

```
16.     inputDirPath = args[0];                          // directory path
17.   else
18.     if (args.length==0)
19.       inputDirPath = System.getProperty("user.dir");  // working directory
20.     else
21.       {
22.       System.out.printf("number of arguments must be 0/1 and not %d\n",
23.                         args.length);
24.       return;           // exit main()
25.       }
26.
27.   File inputDir = new File(inputDirPath);   // open file or directory
28.   String[] inputDirFiles = null;
29.   if (inputDir.isDirectory())               // got a directory?
30.     inputDirFiles = inputDir.list();
31.   else
32.     {
33.     System.out.println("input not a directory: " + inputDirPath);
34.     return;           // exit main()
35.     }
36.
37. // ***
38. // *** (limited) output of directories/files found
```

```
39. // ***
40.   String fileSeparator = System.getProperty("file.separator");
41.                         // is "/" on UNIX and "\" on Windows
42. //
43.   System.out.println(" ");
44.   System.out.println("files and directories found in: " + inputDirPath);
45.   System.out.println(" ");
46.   for (int i=0;i<Math.min(10,inputDirFiles.length);i++)
47.     {
48.     String fullPath = inputDirPath + fileSeparator + inputDirFiles[i];
49.     File listFile = new File(fullPath)        ;   // open file or directory
50.     if (listFile.isFile())                        // got a file?
51.       System.out.println("file: " + fullPath);
52.     else
53.       System.out.println(" dir: " + fullPath);
54.    }
55.
56.   }  // end of DirectoryHandling.main()
57. }    // end of DirectoryHandling
```