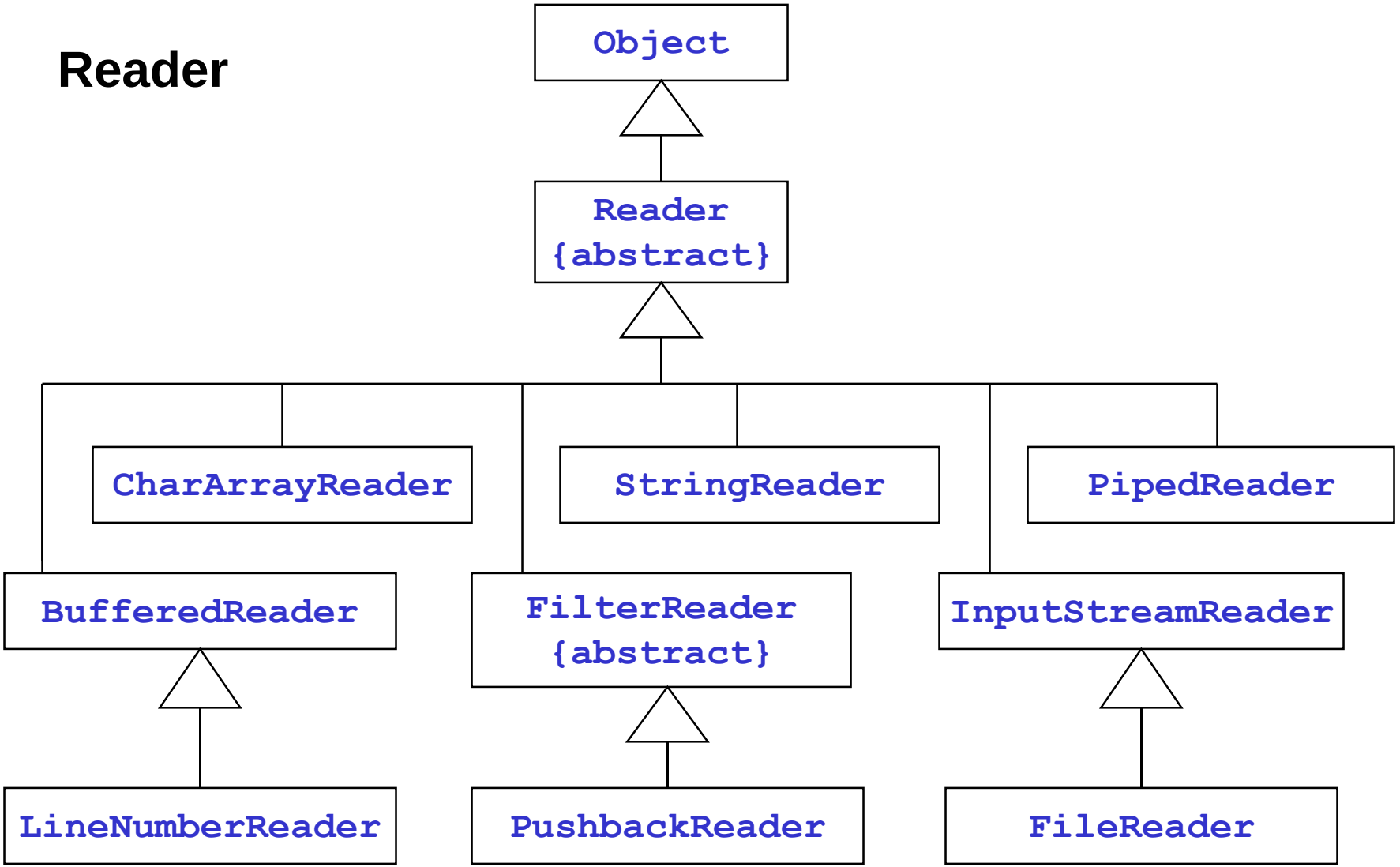
Ein-, Ausgabemodell:

basiert auf Datenströmen (Datenquellen und Datensenken)
Daten: Texte, html-Code, pdf-, mp3-Dateien, …

Datenarten:

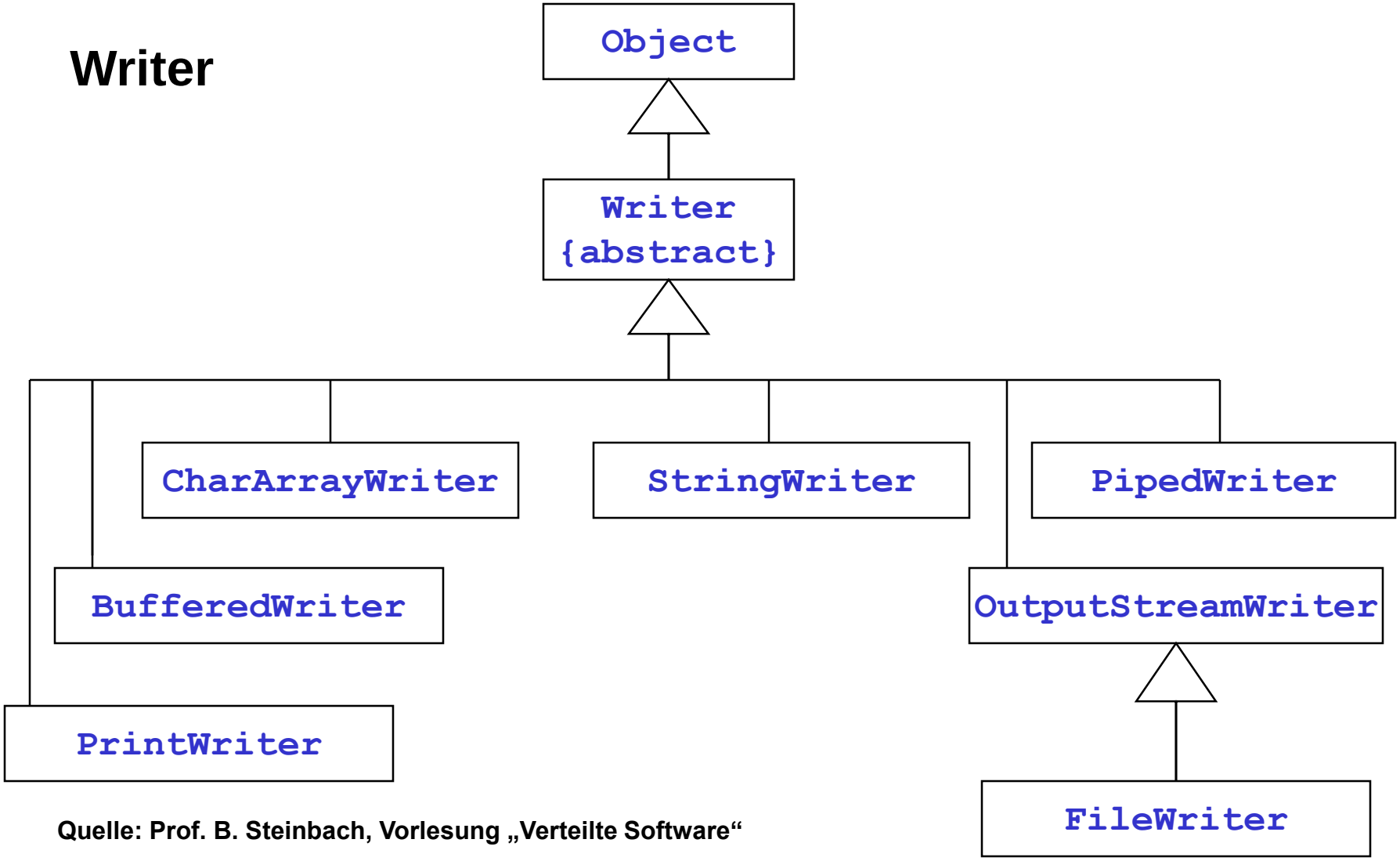
- Zeichen (char) – `Reader, Writer`
- Binärdaten (byte) – `InputStream, Outputstream`

Reader



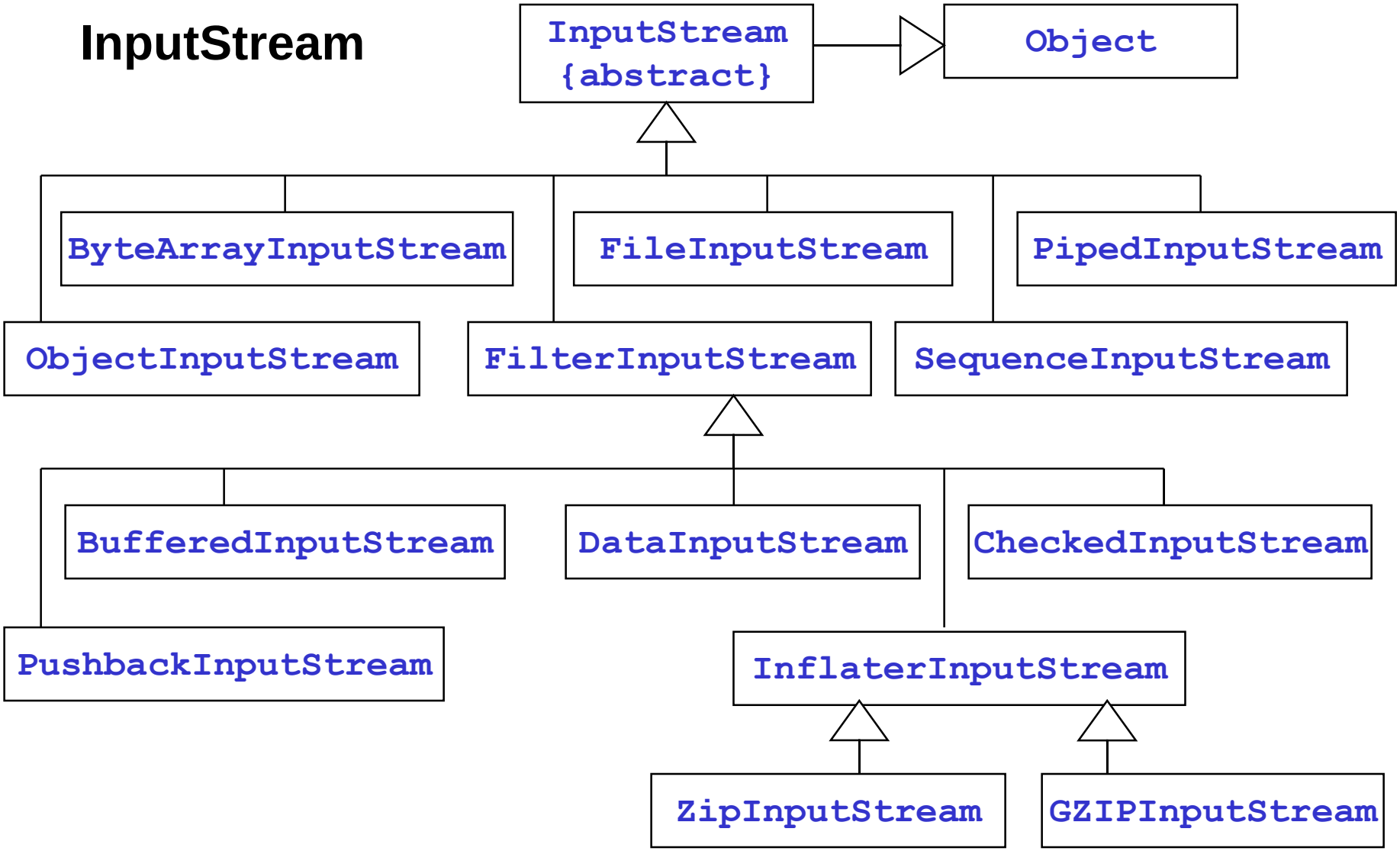
Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

Writer



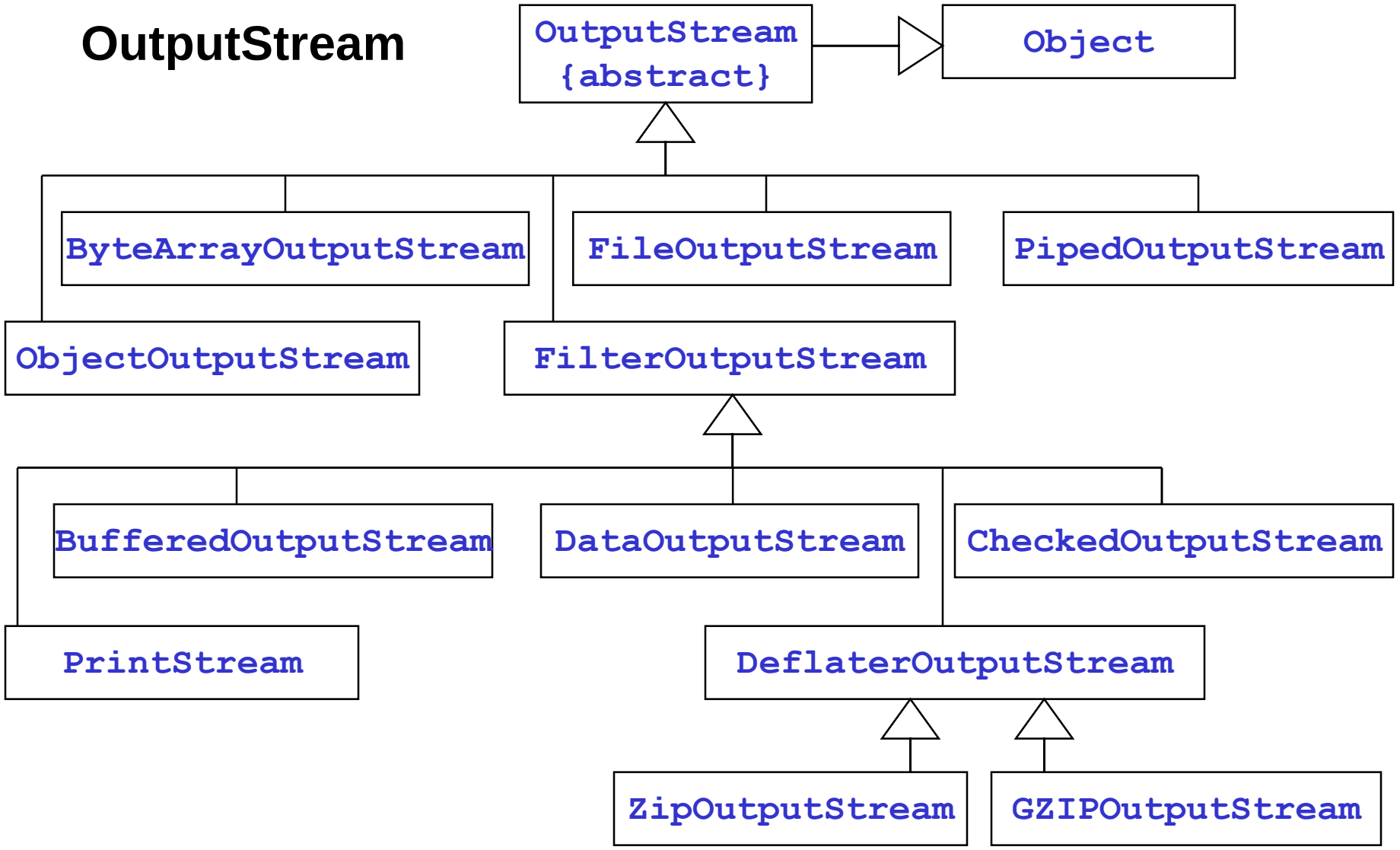
Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

InputStream



Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

OutputStream



Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

Dekorator Muster (decorator pattern)

- eine Möglichkeit die Klasse um die zusätzlichen Funktionalität zu erweitern
- Strukturmuster
- Instanz des Dekorierers wird vor der Instanz zu dekorierenden Klasse geschaltet:

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

- beide haben die gleiche Schnittstelle (Methode, z.B. read, write)

```
int zeichen=br.read();
```

- Aufrufe werden vom Dekorierer an Dekorierten weitergeleitet

Klasse java.lang.System

static-Objekte der Klassen **InputStream**, **PrintStream**:

Standardeingabe	in
Standardausgabe	out
Standardfehlerausgabe	err

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

System.out.println("Ausgabe");
```

Klasse File (java.io)

```
File( String pathname )
File( URI uri )
```

```
File f = new File ("C:/");
System.out.println(f); //Ausgabe nur C:/ kein Inhalt von C
```

Klasse Scanner (java.util.Scanner):

- Zerlegung von Sources in Token, Einlesen Token-weise

- ableleitet von java.util.Iterator<E>
- erbt
  hasNext(): boolean
  next(): E
  remove()

- definiert viele weitere Methoden

```
hasNextByte()
nextByte()
hasNextInt()
nextInt()
hasNextDouble()
nextDouble()
hasNextBoolean()
nextBoolean()
hasNextLong()
nextLong()
```

Konstruktoren:

Scanner(File source)
Scanner(InputStream source)
Scanner(String source)

Beispiele:

```
String text = "Hänsel-und-Gretel\ngingen-durch-den-Wald";
Scanner scanner = new Scanner( text ).useDelimiter( "-" );

//lesen einer datei zeilenweise
import java.io.*;
import java.util.Scanner;

public class ReadAllLines
{
  public static void main( String[] args )
                                throws FileNotFoundException
  {
    Scanner scanner = new Scanner( new File("quelle.txt") );
    while ( scanner.hasNextLine() )
      System.out.println( scanner.nextLine() );
    scanner.close();
  }
}
```

InputStream

```
int read()
int read(byte[] buf)
int read(byte[] buf, int off, int len)
void close()
```

Reader

```
int read()
int read(char[] cbuf)
int read(char[] cbuf, int off, int len)
void close()
```

OutputStream

```
void write(int b)
void write(byte[] b)
void write(byte[] b, int off, int len)
void flush()
void close()
```

- Rückgabe -1, falls das Ende des Streams erreicht wurde,
  sonst Anzahl der gelesenen Bytes (Zeichen)
- blockieren bis Eingabe stattgefunden hat bzw. das Stream-Ende vorliegt
- flush() erzwingt die gepufferte Ausgabe in den Stream
- werfen IOException

```
import java.io.IOException;

public class InpByte {

public static void main(String[] args) {
 byte bi,
      b[]  = new byte[10];
 try
 {
     bi = (byte) System.in.read();
     System.out.println(bi);//achtung: byte
     System.out.flush();
     int anz = System.in.read(b);

     for (int k = 0; k < anz; k++)
        System.out.println("Zeichen [" + k + "] " + b[k] );
 }
 catch (IOException e) { System.out.println (e); }
 }
}
```

```
a abcde
97
Zeichen [0] 32
Zeichen [1] 97
Zeichen [2] 98
Zeichen [3] 99
Zeichen [4] 100
Zeichen [5] 101
Zeichen [6] 13
Zeichen [7] 10
```

Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

```
import java.io.*;

public class InpChar {
   public static void main (String args[]) {
      BufferedReader in
         = new BufferedReader(
           new InputStreamReader(System.in));
      int  anz;
      char c,
           cf[] = new char[10];
      try
      {
        c = (char) in.read();
        System.out.write(c);
        System.out.flush();
        System.out.println();
        anz = in.read(cf);
        for (int k = 0; k < anz; k++)
           System.out.println("Zeichen [" + k + "]  "
               + (byte)cf[k] + " - "+ cf[k]);
      }
      catch (IOException e) { System.out.println (e); }
   }
}
```

```
a abcde
a
Zeichen [0]  32 -
Zeichen [1]  97 - a
Zeichen [2]  98 - b
Zeichen [3]  99 - c
Zeichen [4]  100 - d
Zeichen [5]  101 - e
Zeichen [6]  13 -

Zeichen [7]  10 -
```

Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

```
public class ReadFile{
public static void main (String args[])
  {
     String DateiName = getFileName();
     FileInputStream f = null;
     try {
          f = new FileInputStream (DateiName);
          int zeichen;
          while (( zeichen = f. read() ) != -1) System.out.write(zeichen);
          System.out.flush();
          f.close();
     }
     catch (FileNotFoundException io){
          System.out.println ("Datei nicht gefunden.");
     }
     catch (IOException e) {
          System.out.println("Datei "+DateiName+" nicht lesbar.");
     }
  }

  static String getFileName(){
     Scanner data = new Scanner(System.in);
     String dateiname=data.next();
     data.close();
     return dateiname;
  }
}
```

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;
```

Quelle: Prof. B. Steinbach, Vorlesung „Verteilte Software“

PrintStream

- ermöglicht Ausgabe einfacher Datentypen im Klartext in einen Stream
- kein Auswerfen von Exception, sondern Abfrage eines internen Flags möglich
- optional formatierte Ausgabe

Konstruktoren der Klasse:

```
PrintStream(OutputStream out)

PrintStream(OutputStream out, boolean autoFlush)

PrintStream(String fileName) throws FileNotFoundException

PrintStream(File file) throws FileNotFoundException
```

Methoden der Klasse:

```
public void write(int b)                                    //ohne formatierung
public void write(byte[] buf, int off, int len)
public void print(boolean b)                                //verschiedene datentypen
public void print(char c)
public void print(int i)
public void print(long l)
public void print(float f)
public void print(double d)
public void print(char[] s)
public void print(String s)
public void print(Object obj)
public void println()                                       //mit zeilenumbruch
public void println(boolean x)
...                                                         //formatierte ausgabe
public PrintStream printf(Locale l, String format, Object... args)
public PrintStream printf(String format, Object... args)
public PrintStream format(Locale l, String format, Object... args)
public PrintStream format(String format, Object... args)
public boolean checkError()//false beim erfolgreichen Schreiben
public void flush()
public void close()
```