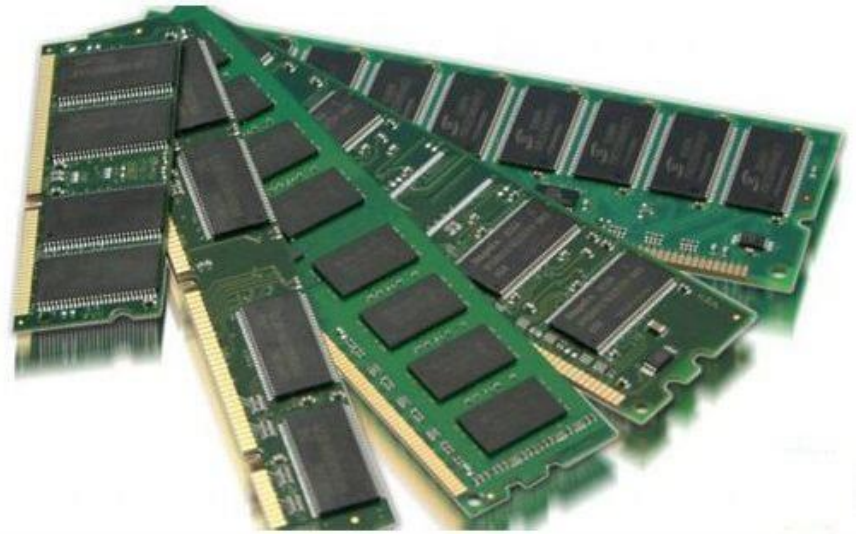
- DA343A -
Objektorienterad programutveckling, trådar och datakommunikation

Föreläsning 6

Johan Holmgren (Utvecklad av Fabian Lorig)



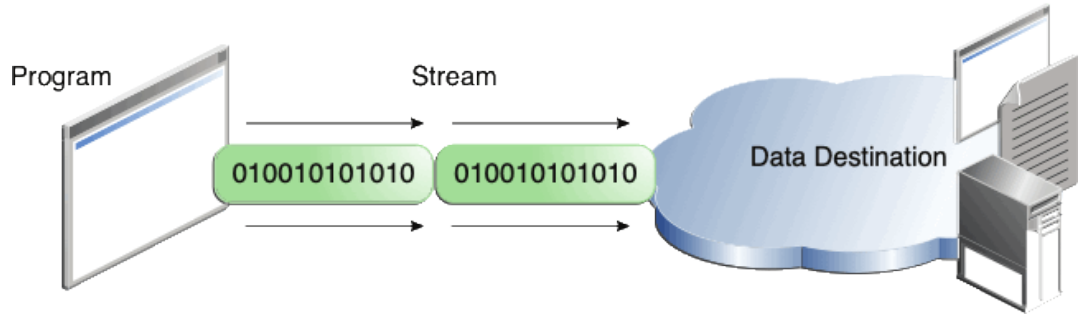
Spara och ladda data

- Hittills har alla beräkningar varit ”flyktiga”
  - Data försvinner när exekveringen av programmet avslutar / blir avslutad
- Olika typer av datorminne:
  - Arbetsminnen (snabb men flyktig; innehåll går förlorat när vi stänger av datorn)
  - Lagringsminnen (långsammare men beständig; lagra data även om datorn blir avstängd)
- Genom att kunna spara och ladda data kan mer avancerade applikationer utvecklas

Att läsa eller skriva data: Strömmar

Man har ofta behov av att flytta data mellan t.ex.

- program och hårddisk
- två program på samma dator
- olika datorer

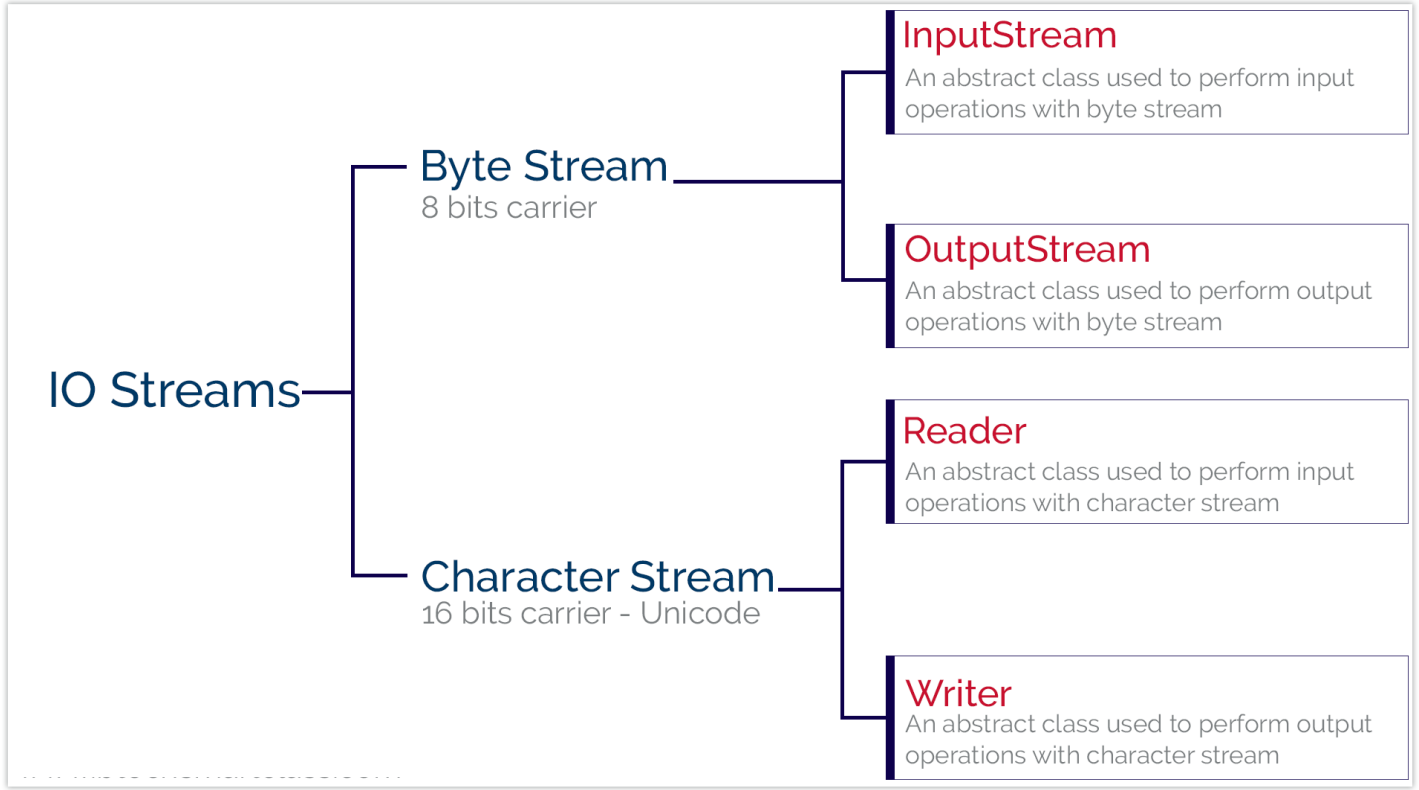


Strömmar är dataflöden (sekventiell följd av tecken), ofta mellan programmet och olika enheter, t.ex. en fil på hårddisken eller ett program på en annan dator.

Om flödet ska gå i båda riktningarna krävs det två strömmar. Flödet från programmet kallas för **utström** (”att skriva data”) och flödet till programmet för **inström** (”att läsa data”).

Java.io

- API som kan användas för att läsa eller skriva data (input and output)
- Finns olika klasser för olika typer av data
- Tecken-baserade klasser: ***Reader, Writer***
- Byte-baserade klasser: ***InputStream, OutputStream***



Java.io

	Byte Based		Character Based	
	Input	Output	Input	Output
Basic	InputStream	OutputStream	Reader InputStreamReader	Writer OutputStreamWriter
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	FileInputStream RandomAccessFile	FileOutputStream RandomAccessFile	FileReader	FileWriter
Pipes	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
Buffering	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
Filtering	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
Parsing	PushbackInputStream StreamTokenizer		PushbackReader LineNumberReader	
Strings			StringReader	StringWriter
Data	DataInputStream	DataOutputStream		
Data - Formatted		PrintStream		PrintWriter
Objects	ObjectInputStream	ObjectOutputStream		
Utilities	SequenceInputStream			

Unicode

- Samling av skrivtecken (bokstäver, siffror, matematiska symboler, …)
- Teckenkodning behövs för att representera Unicodetext
- T.ex. UTF-16 (*16 bitars unicode transformations-format*), som använder sig av sekvenser av dubbel-oktetter (bytes) för att lagra tecknen i datorer
- Standard kodning i Java är UTF-16 av unicode
- Finns flera andra kodningar, t.ex. UTF-8, ASCII, …

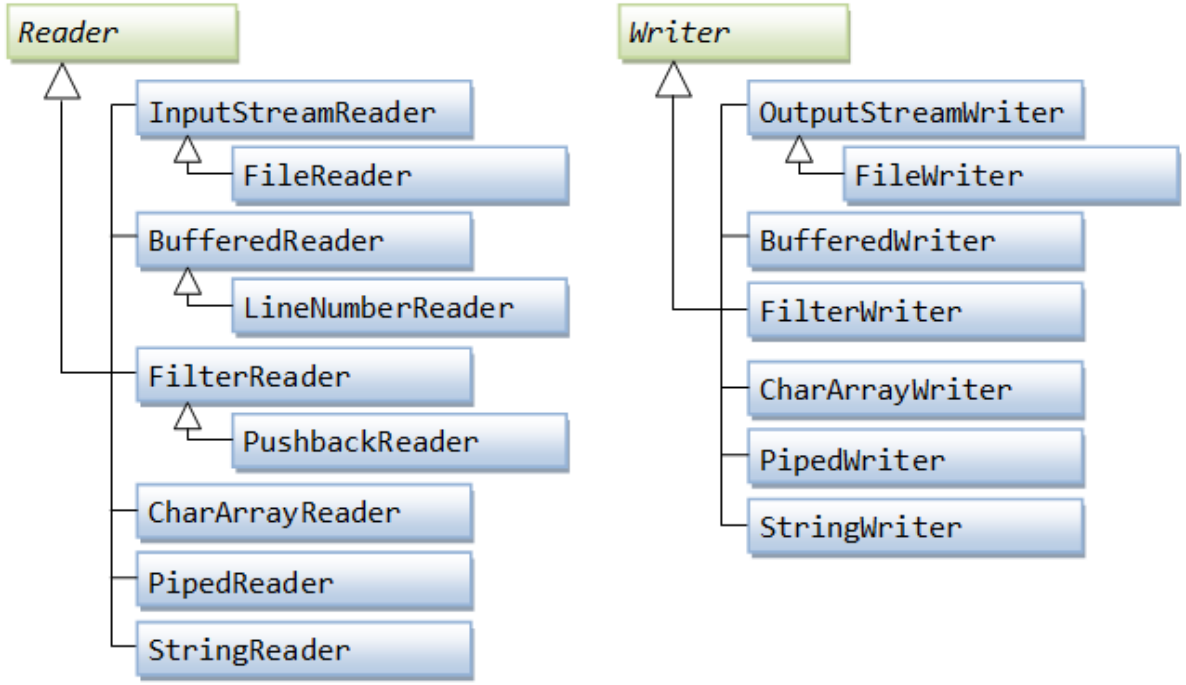
Graphic character symbol	Hexadecimal character value

0020	0 0030	@ 0040	P 0050	` 0060	p 0070	00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
" 0022	2 0032	B 0042	R 0052	b 0062	r 0072	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
' 0027	7 0037	G 0047	W 0057	g 0067	w 0077	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
(0028	8 0038	H 0048	X 0058	h 0068	x 0078	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
- 002D	= 003D	M 004D	] 005D	m 006D	} 007D	- 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
/ 002F	? 003F	O 004F	_ 005F	o 006F	007F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

	A	א	好	不
Code point	U+0041	U+05D0	U+597D	U+233B4
UTF-8	41	D7 90	E5 A5 BD	F0 A3 8E B4
UTF-16	00 41	05 D0	59 7D	D8 4C DF B4
UTF-32	00 00 00 41	00 00 05 D0	00 00 59 7D	00 02 33 B4

Textströmmar

- Grundläggande klasser för att läsa och skriva text (unicode-tecken) är ***Reader*** och ***Writer***. (se föregående slides)
- Båda klasserna är abstrakta



Textströmmar (skriva)

De flesta metoder kan kasta undantag av typen *IOException*.

Writer-klasser ärver den abstrakta klassen ***Writer***, vilken innehåller ett antal metoder, bl.a.

public void close()
Stänger strömmen.

public void flush()
Bufferten överförs till målet.

public void write(int c)
Skriver tecknet **c** till strömmen.

public void write(char[] c *eller* String s)
Skriver **c** eller **s** till strömmen.

public void write(String s, int start, int len)
Skriver **s** till strömmen, från position **start** och **len** tecken.

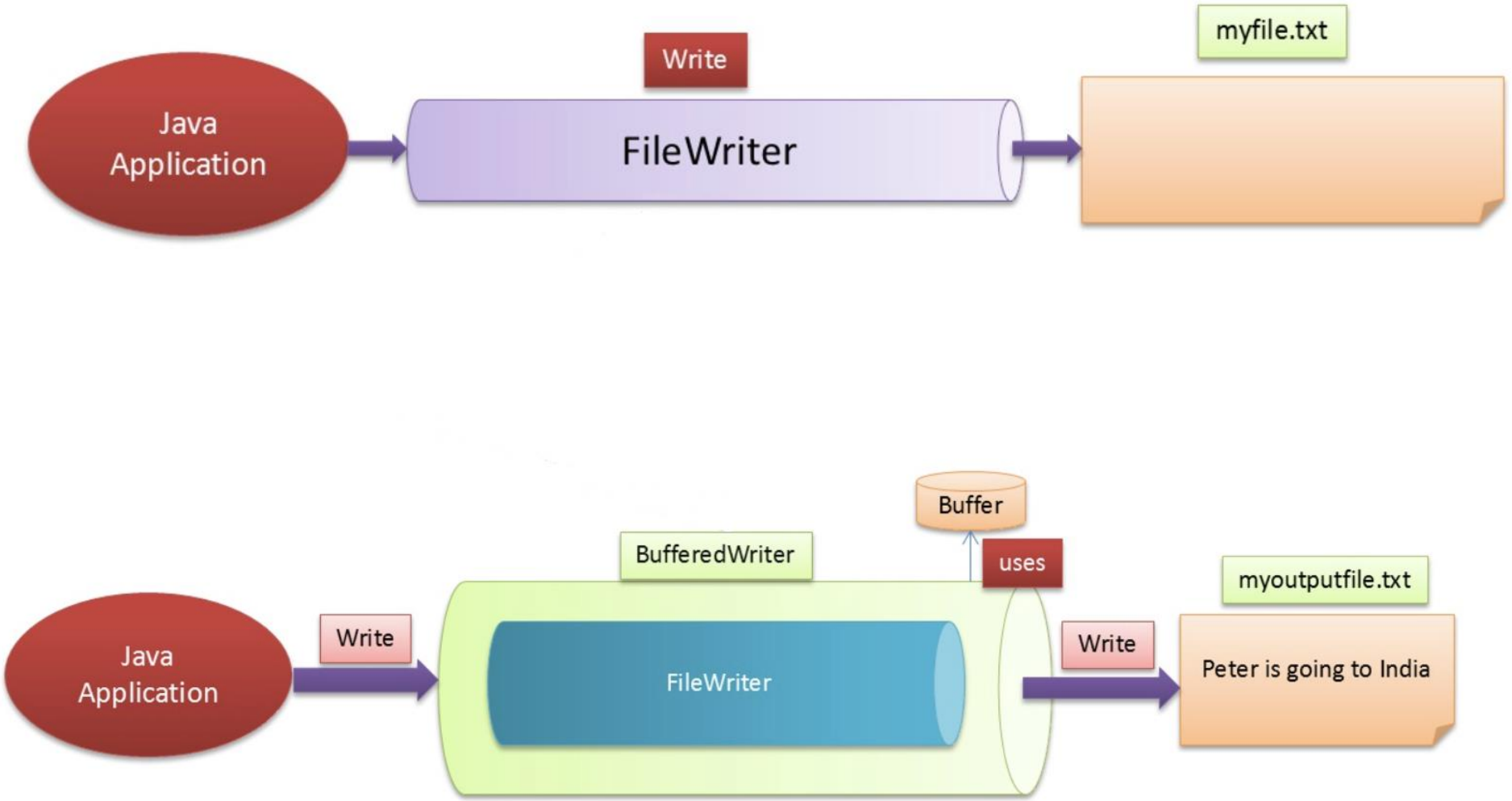
Några intressanta ***Writer***-subklasser är:

BufferedWriter
Skriver med hjälp av buffert. Innehåller bl.a. metoden
public void newLine() - skriver radslutstecken till strömmen.

OutputStreamWriter
Skriver till en ***OutputStream***.
Kan användas för att ange teckenkodning vid skrivning till en textfil.

FileWriter
Skriver till en fil på hårddisken

Writer



Textströmmar (läsa)

De flesta metoder kan kasta undantag av typen *IOException*.

Reader-klasser ärver den abstrakta klassen ***Reader***, vilket innehåller ett antal metoder, bl.a.

public void close()
Stänger strömmen.

public int read()
Läser ett tecken. Returnerar -1 om strömmen inte innehåller fler tecken.

public int read(char[] c)
Läser tillgängliga tecken.
Returnerar antal lästa tecken / -1

public abstract int read(char[] c, int start, int len)
Läser tillgängliga tecken, men aldrig fler än **len**.
Lästa tecken placeras i **c** med början i position **start**.
Returnerar antalet lästa bytes / -1.

Några intressanta ***Reader***-subklasser är:

BufferedReader
Läser med hjälp av en buffert. Innehåller bl.a. metoden
public String readLine() – läsa en rad med tecken från strömmen.

InputStreamReader
Läser från en ***InputStream***. Kan användas för att ange teckenkodning vid läsning från en textfil.

FileReader
Läser från en fil på hårddisken.

Exempel: Skriva text till och läsa text från fil på hårddisk

- **Reader** läser en char i taget (inte en byte som InputStream)

char i Java
16 bit Unicode character
från 0 till 65 535
UTF-16

```
Reader reader = new FileReader( fileName: "c:\\data\\file-input.txt");

int data = reader.read();
while(data != -1){
    char dataChar = (char) data;
    data = reader.read();
}
```

```
Writer writer = new FileWriter( fileName: "c:\\data\\file-output.txt");

writer.write( str: "Hello World Writer");
writer.close();
```

DemoReaderWriter1

BufferedReader

- Olika ***Reader*** and ***Writer*** kan kombineras med varandra
- Buffering kan snabba på IO operationer
- Istället för att läsa en char i taget från ***Reader*** läser ***BufferedReader*** en större block

- ***BufferedReader*** har metoden ***readLine()*** som kan användas för att läsa en hel rad
- Efter hela texten lästes in ska ***Reader*** stängas genom metoden ***close()***

```
BufferedReader bufferedReader =
        new BufferedReader(
                new FileReader( fileName: "c:\\data\\file-input.txt"));

String line = bufferedReader.readLine();
while(line != null) {
    System.out.println(line);
    line = bufferedReader.readLine();
}
bufferedReader.close();
```

DemoReaderWriter1

BufferedWriter

- Samma som ***BufferedReader*** men för att kunna skriva text till en fil
- Kan skriva en char i taget eller en array av chars
- *Obs!* Metoden ***flush()*** används för att säkerställa att all data som sparades i ***BufferedWriter*** skickades till ***FileWriter***
- ***close()*** behövs inte om ***Writer*** öppnades i en ***try-catch***

```
FileWriter output = new FileWriter( fileName: "c:\\data\\file-output.txt");

try(BufferedWriter bufferedWriter =
            new BufferedWriter(output)){

    bufferedWriter.write( str: "Hello World");

}
```

DemoReaderWriter2

Konvertering av data

- Oftast vill man skriva text (tecken) i en annan kodning i en fil (bytes)
- Möjlighet att kombinera ***Stream*** och ***Writer***

- ***OutputStreamWriter(OutputStream out, String encoding)*** t.ex. med encoding ”ISO-8859-1” eller ”UTF-8”
- Tar hand om konvertering från tecken till bytes

- ***FileOutputStream*** istället för ***FileWriter***

```
try { bw = new BufferedWriter(
            new OutputStreamWriter(
                    new FileOutputStream( name: "c:\\data\\file-output.txt"),
                    charsetName: "ISO-8859-1")
     );
    bw.write( str: "hello world" );
    bw.flush();
} catch(IOException e) {
    System.err.println(e);
} finally {
    if(bw != null) {
        try {
            bw.close();
        } catch(IOException e){}
    }
}
```

DemoReaderWriter2

Läsa text från fil på hårddisk

- Att läsa text i form av bytes från en fil med konvertering till tecken fungerar likadant

- Om man önskar ange teckenkodning för textfilen använder man ***FileInputStream*** tillsammans med ***InputStreamReader***:

```
FileInputStream fis = new FileInputStream(”ex.txt");

InputStreamReader isr = new InputStreamReader( fis,
                                "ISO-8859-1" );
```

- För att effektivisera läsningen så kopplar man strömmen till en buffrad ström:

```
BufferedReader br = new BufferedReader( isr );
```

```
try(BufferedReader br = new BufferedReader(
        new InputStreamReader(
            new FileInputStream( name: "c:\\data\\file-input.txt"),
            charsetName: "ISO-8859-1")
        )
)
{
    line = br.readLine();
    while(line != null) {
        System.out.println(line);
        line = br.readLine();
    }
}   catch(IOException e) {
    System.err.println( e );
}
```

DemoReaderWriter3

Skriva rader med text till fil på hårddisk

Man skapar en ström till en fil för att skriva data:

```
FileOutputStream fw = new FileOutputStream(”temp/ex.txt");

OutputStreamWriter osw = new OutputStreamWriter( fw, "UTF-8" );
```

För att effektivisera skrivningen så kopplar man strömmen till en buffrad ström:

```
BufferedWriter bw = new BufferedWriter( osw );
```

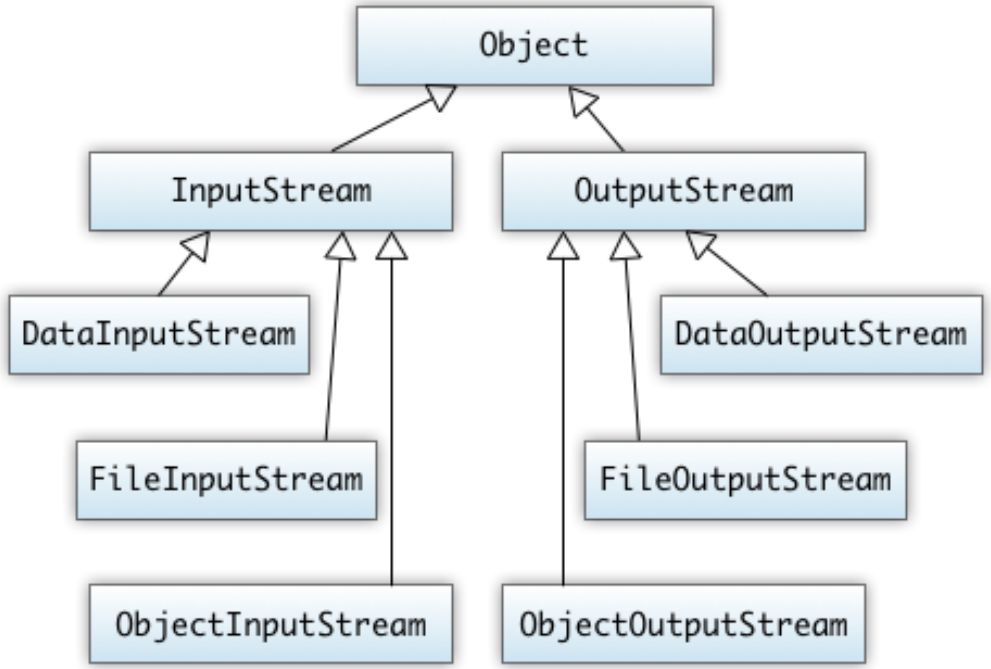
Skrivning sker ofta med metoderna

```
write(String), newLine() och flush()
```

```
public void write(String str) {
   String[] rows;

   rows = str.split(“\n”);

   for(String s : rows) {
      bw.write(s);     // skriv rad
      bw.newLine();              // skriv radslut
      bw.flush();      // för over till hårddisken
   }

}
```

Dataströmmar (streams)

- Grundläggande (abstrakta) klasser för att läsa och skriva bytes är ***InputStream*** och ***OutputStream***
- 8-bit (bytes) istället för 16-bit unicode (tecken)



Output-strömmar

Output-strömmar ärver klassen ***OutputStream*** vilken innehåller metoderna:

`public abstract void write( int b )`
Skriver en byte

`public void write( byte[] data )`
Skriver en array med bytes

`public void write( byte[] data, int start, int len )`
Skriver len bytes med början i positionen *start*

`public void close()`
Stänger strömmen

`public void flush()`
Tömmer buffertar

Output-strömmarna kan delas upp i två kategorier

1. De vars syfte är att skriva till ett speciellt mål, t.ex.:

ByteArrayOutputStream	Skriva till byte-array
FileOutputStream	Skriva till hårddisk
PipedOutputStream	Skriva till en tråd

2. De vars syfte är att ändra innehållet i strömmen, t.ex.:

BufferedOutputStream	Skriva med hjälp av buffert
DataOutputStream	Skriva olika datatyper
ObjectOutputStream	Skriva objekt
ZipOutputStream	Skriva komprimerad data
CipherOutputStream	Skriva krypterad data

Output-strömmar

DataOutputStream och ***ObjectOutputStream*** implementerar interfacet ***DataOutput*** och innehåller därmed bl.a. dessa metoder:

- **`public void writeBoolean(boolean)`** Skriva en boolean
- **`public void writeByte(byte)`** Skriva en byte
- **`public void writeChar(char)`** Skriva en char
- **`public void writeChars(String)`** Skriva en sträng
- **`public void writeDouble(double)`** Skriva en double
- **`public void writeFloat(float)`** Skriva en float
- **`public void writeInt(int)`** Skriva en int
- **`public void writeLong(long)`** Skriva en long
- **`public void writeShort(short)`** Skriva en short
- **`public void writeUTF(String)`** Skriva en UTF-kodad sträng

ObjectOutputStream har metoder för att skriva objekt, t.ex.:

- **`public void writeObject(Object obj)`** Skriva ett objekt

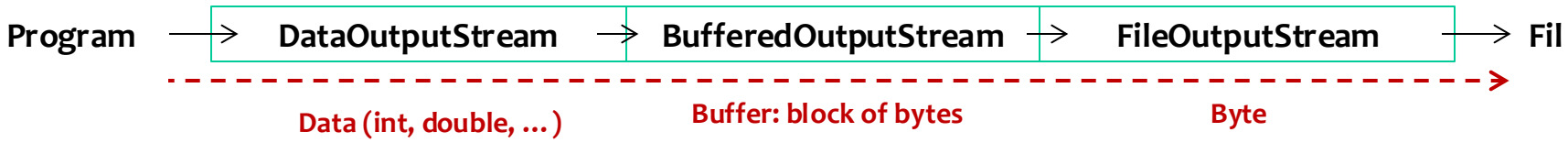
Strömmar – kedjas ihop

Obs! Eftersom vi skriver utan teckenkodning går .dat-filen inte att läsa för människor.

Strömmarna, vars syfte är att ändra strömmens innehåll, kan kedjas ihop.

Exempel på kedja:

```
FileOutputStream fos = new FileOutputStream(”files/stats.dat”);
BufferedOutputStream bos = new BufferedOutputStream(fos);
DataOutputStream dos = new DataOutputStream(bos);
```



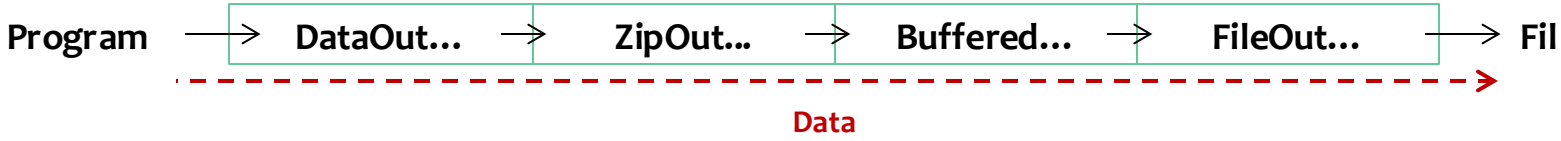
Strömmar – kedjas ihop

Strömmarna, vars syfte är att ändra strömmens innehåll, kan kedjas ihop.

Exempel på kedja:

```
FileOutputStream fos = new FileOutputStream(”files/stats.dat”);
BufferedOutputStream bos = new BufferedOutputStream(fos);
ZipOutputStream zos = new ZipOutputStream(bos);
DataOutputStream dos = new DataOutputStream(zos);
```

En ström i vilken man kan skriva data i komprimerat format till en fil. Buffringen effektiviserar skrivningen.



Skriva enkla datatyper

WriteReadData1

WriteReadData2

```
FileOutputStream fos = null;
BufferedOutputStream bos = null;
DataOutputStream dos = null;
```

1. Skapa en ström för att skriva till en fil:

```
try {
   fos = new FileOutputStream(”temp/ex.dat”);
```

2. För att effektivisera kan skrivningen buffras

```
   bos = new BufferedOutputStream(fos);
```

3. För att skriva enkla datatyper så kopplar man strömmen till ett objekt av typen ***DataOutputStream***:

```
dos = new DataOutputStream(bos);
```

4. Nu kan man skriva data till strömmen och filen

```
dos.writeUTF(”HEJ”);
dos.writeInt(1000);
dos.flush();
```

5. När man skrivit färdigt ska strömmarna stängas genom anrop till **close**-metoden.

   Det räcker att stänga den yttersta strömmen om allt gått som det ska:

```
} finally {
    try {
        dos.close();
    } catch(Exception e) {}
}
```

Dispose pattern (try with resources)

För att säkerställa att en ström stängs används en teknik kallad ***dispose pattern***:

```
DataOutputStream dos = null;
try {
    dos = new DataOutputStream( new BufferedOutputStream(
              new FileOutputStream( filename )));
         // använda strömmen
} catch(IOException e) {
    System.err.println(e);
} finally {
    if(dos != null) {
        try {
            dos.close();
        } catch(IOException e){}
    }
}
```

Ett enklare alternativ är att använda en speciell try-sats, den s.k. ***try with resources***

```
try(DataOutputStream dos = new DataOutputStream(
                               new BufferedOutputStream(
                                  new FileOutputStream( filename )))  {


      // använda strömmen


} catch(IOException e) {
    System.err.println(e);
}
```

Input-strömmar

Input-strömmar ärver klassen ***InputStream*** vilken bl.a. innehåller metoderna

`public abstract int read()`
Läser en byte. Returnerar -1 om EOF

`public int read(byte[] data)`
Läser bytes till byte-array.
Returnerar antalet lästa bytes.
Returnerar -1 om EOF.

`public int read(byte[] data, int start, int len)`
Läser bytes (max *len* st) till byte-array med start i positionen *start*.
Returnerar antalet lästa bytes / -1

`public void close()`
Stänger strömmen.

Input-strömmarna kan delas upp i två kategorier:

1. De vars syfte är att läsa från en speciell källa, t.ex.:
   ByteArrayInputStream Läsa från byte-array
   FileInputStream Läsa från hårddisk
   PipedInputStream Läsa från en tråd

2. De vars syfte är att ändra innehållet i strömmen, t.ex.:
   BufferedInputStream Läsa med hjälp av buffert
   DataIntputStream Läsa olika datatyper och strängar
   ObjectInputStream Läsa objekt
   ZipInputStream Läsa komprimerad data
   CipherInputStream Läsa krypterad data

Input-strömmar

DataInputStream och ***ObjectInputStream*** implementerar interfacet ***DataInput*** och innehåller därmed bl.a. dessa metoder:

- **public boolean readBoolean()** Läsa en boolean
- **public byte readByte()** Läsa en byte
- **public char readChar()** Läsa en char
- **public double readDouble()** Läsa en double
- **public float readFloat()** Läsa en float
- **public int readInt()** Läsa en int
- **public long readLong()** Läsa en long
- **public short readShort()** Läsa en short
- **public String readUTF()** Läsa en UTF-kodad sträng

ObjectInputStream har metoder för att läsa objekt, t.ex.

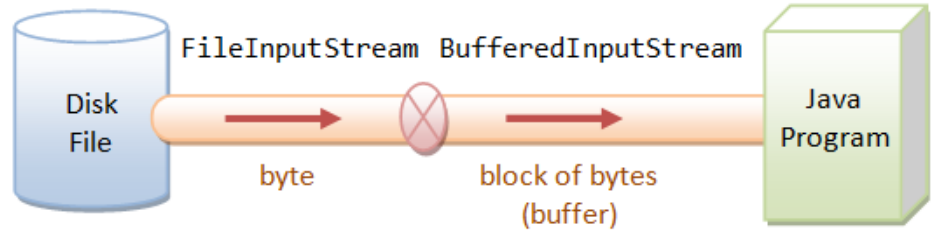
- **public Object readObject()** Läsa ett objekt

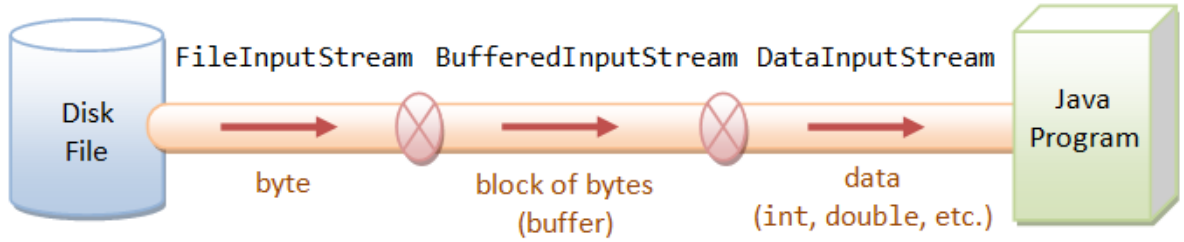
Strömmar – kedjas ihop

Strömmarna, vars syfte är att ändra strömmens innehåll, kan kedjas ihop.

Exempel på kedja:

```
DataInputStream dis = new DataInputStream(
    new ZipInputStream(
        new BufferedInputStream(
            new FileInputStream( name: "files/stats.dat")
        )
    )
);
```





Läsa enkla datatyper

WriteReadData1

WriteReadData2

Man skapar en ström för att läsa från en fil.

För att effektivisera buffrar man läsningen.

1. För att läsa enkla datatyper så kopplar man strömmen till ett objekt av typen **DataInputStream**.

```
DataInputStream  dis = new DatainputStream(
  new BufferedInputStream(
       new FileInputStream(”temp/ex.dat”)
  )
);
```

2. Därefter kan man läsa data från filen.

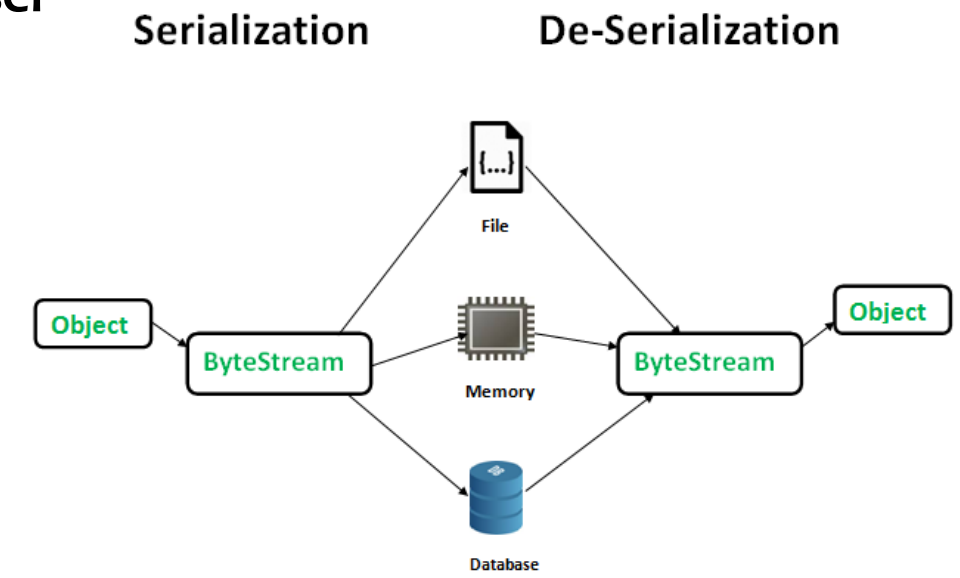
```
str = dis.readUTF();
nbr = dis.readInt(1000);
```

3. När man läst färdigt ska strömmarna stängas genom anrop till **close**-metoden. Det räcker att stänga den yttersta strömmen om allt gått som det ska:

```
} finally {
    try {
        dis.close();
    } catch(Exception e) {}
}
```

Skriva och läsa ett objekt: Serialisering

- För att skriva ett objekt måste **samtliga fält i objektet** och **alla objekt som det skrivna objektet innehåller referenser till** skrivas
- Serialisering: överföra tillståndet av ett objekt till byte-ström (motsats: deserialisering)
- ***ObjectOutputStream*** och ***ObjectInputStream*** (alltid binärströmmar, finns inga reader/writer)
- Metoderna *writeObject*() och *readObject*() som läser/skriver ett helt objekt
- Klassen måste implementera interfacet *Serializable*



Interface Serializable

- Innehåller inga metoder
- **SerialVersionUID:** används vid deserialisering för att kontrollera om objektet skapades med samma version av klassen
- UID ska ändras när klassen ändras

```
private static final long serialVersionUID = 123456L
```

Serializable

Person.java

PersonTest.java

Klassen Person implementerar *Serializable* varvid Person-instanser kan användas tillsammans med strömmar.

```
import java.io.*;
class Person implements Serializable {
   private String name;
   private Person partner;
```

Skriva Person-array

```
private static void writePersons(Person[] pers) throws IOException {
    try (ObjectOutputStream oos = new ObjectOutputStream(
                                  new FileOutputStream("files/personer.dat"))){
        oos.writeObject(pers);
    }
}
```

Serializable

Person.java

PersonTest.java

Klassen Person implementerar *Serializable* varvid Person-instanser kan användas tillsammans med strömmar.

```
import java.io.*;
class Person implements Serializable {
   private String name;
   private Person partner;
```

Läsa Person-array

```
private static Person[] readPersons() throws IOException {
    Person[] pers=null;
    try(ObjectInputStream ois = new ObjectInputStream(
                                new FileInputStream("files/personer.dat"))) {
        pers = (Person[])ois.readObject();
    }
    catch(ClassNotFoundException e) {}
    return pers;
}
```