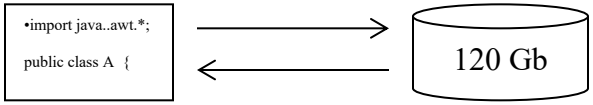
Föreläsning 3

- Strömmar
- Filhantering

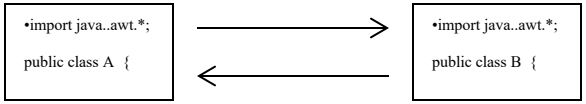
Strömmar

Man har ofta behov av att flytta data mellan

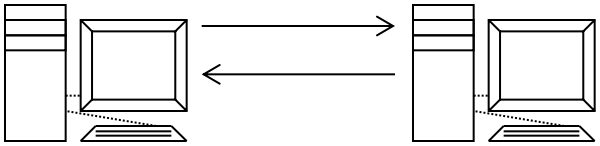
- program och hårddisk



- två program
  - på samma dator



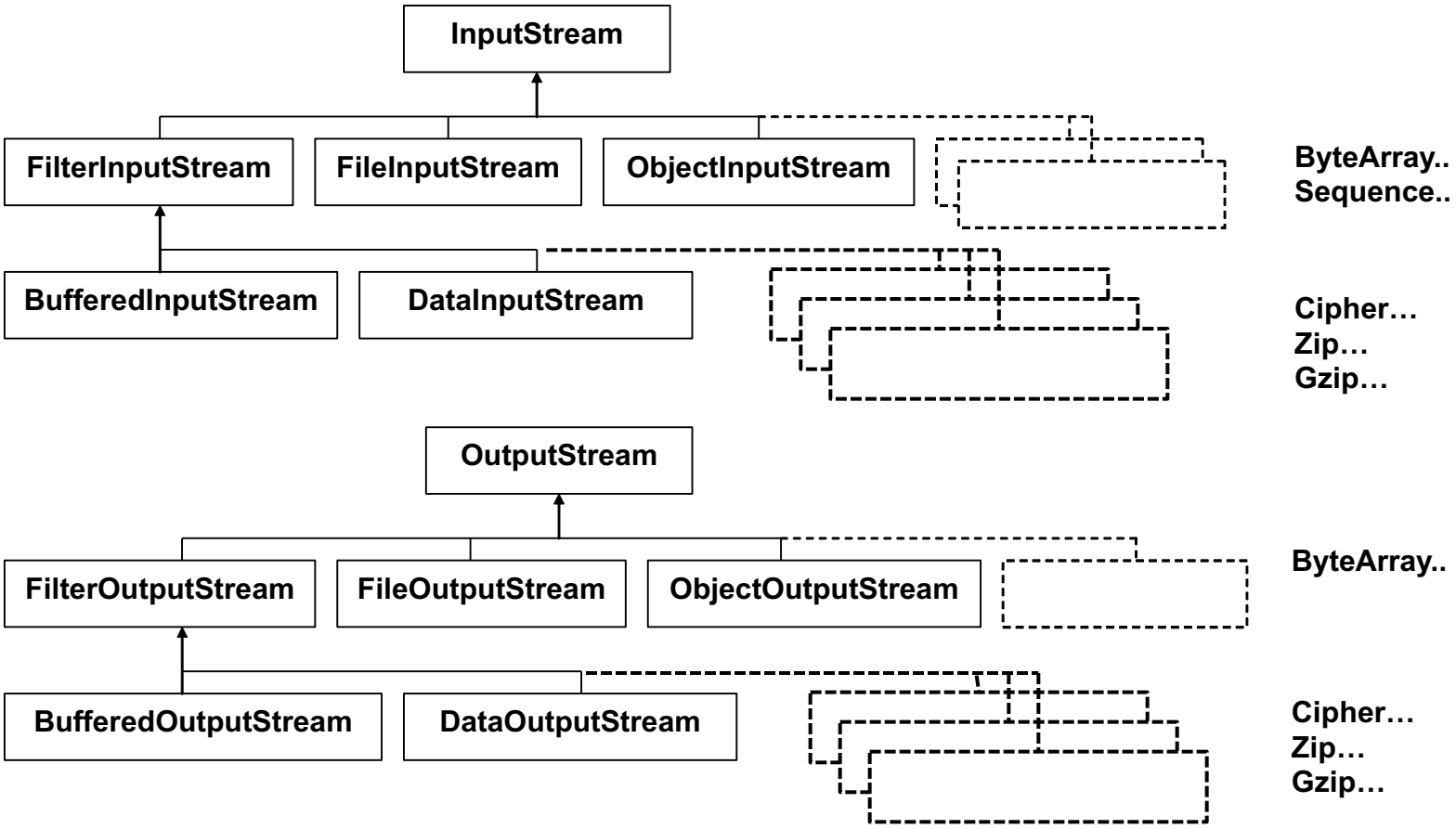
  - på olika datorer



Strömmar är dataflöden mellan källa och mål, 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”).

Dataströmmar

Grundläggande klasser för att beskriva dataströmmar är **InputStream** och **OutputStream**. Båda klasserna är abstrakta.



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 katergorier:

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.:

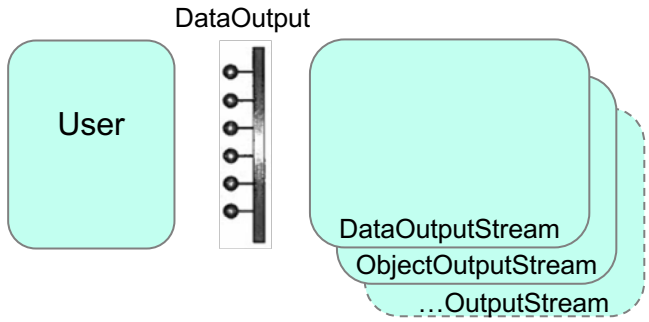
   BufferedOutputStream Skriva med hjälp av buffert
   DataOutputStream Skriva olika datatyper och strängar
   ObjectOutputStream Skriva objekt
   ZipOutputStream Skriva komprimerad data
   CipherOutputStream Skriva krypterad data

Output-strömmar

Ett par output-strömmar i kategori 2 är av speciellt intresse:

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

Samtliga ovanstående metoder kan kasta undantag av typen *IOException*.

Strömmar – kedjas ihop

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

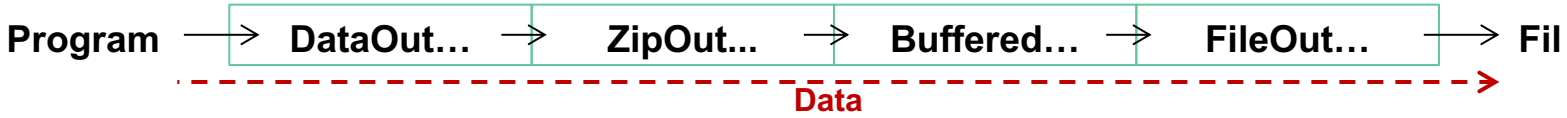
Exempel på kedja:

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

En ström i vilken man kan skriva enkla datatyper, och lite till, till en fil. Buffringen effektiviserar skrivningen.

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

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

Man skapar en ström för att skriva till en fil:

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

För att effektivisera buffrar man skrivningen

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

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

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

Därefter kan man skriva data till filen.

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

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) {}
}
```

WriteReadData1

WriteReadData2

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){}
   }
}
```

Som klassmetod

```
public static void close(Closeable c) {
   try {
      c.close();
   } catch(Exception e) {}
}
```

Ett enklare alternativ är att använda en speciell try-sats, den s.k. ***try with resources*** (fr.o.m. Java 7)

```
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 katergorier:

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

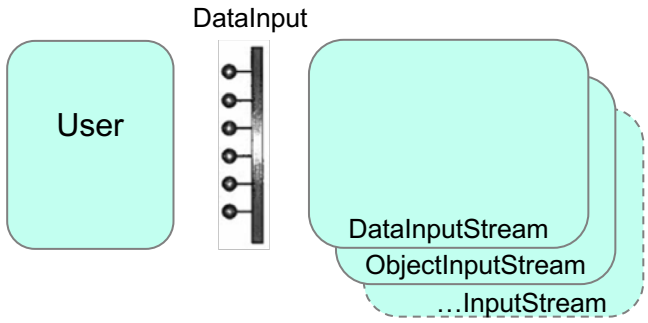
2. De vars syfte är att ändra innehållet i strömmen, t.ex.:
   - ***BufferedInputStream*** — Läsa med hjälp av buffert
   - ***DataIntputStream*** — Läsa olika datatyper och strängar
   - ***ObjectInputStream*** — Läsa objekt
   - ***ZipInputStream*** — Läsa komprimerad data
   - ***CipherInputStream***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

Input-strömmar

Ett par input-strömmar i kategori 2 är av speciellt intresse:

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

Samtliga ovanstående metoder kan kasta undantag av typen *IOException*.

Strömmar – kedjas ihop

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

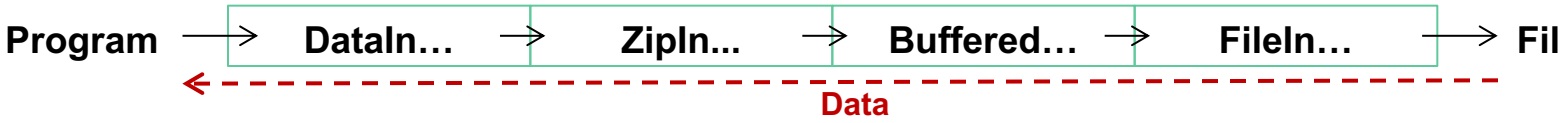
Exempel på kedja:

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

En ström i vilken man kan läsa enkla datatyper, och lite till, från en fil. Buffringen effektiviserar läsningen.

Exempel på kedja:

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



En ström i vilken man kan läsa data i komprimerat format från en fil. Buffringen effektiviserar läsningen.

Läsa enkla datatyper

Man skapar en ström för att läsa från en fil. För att effektivisera buffrar man läsningen. 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”)));
```

Därefter kan man läsadata från filen.

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

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) {}
}
```

WriteReadData1

WriteReadData2

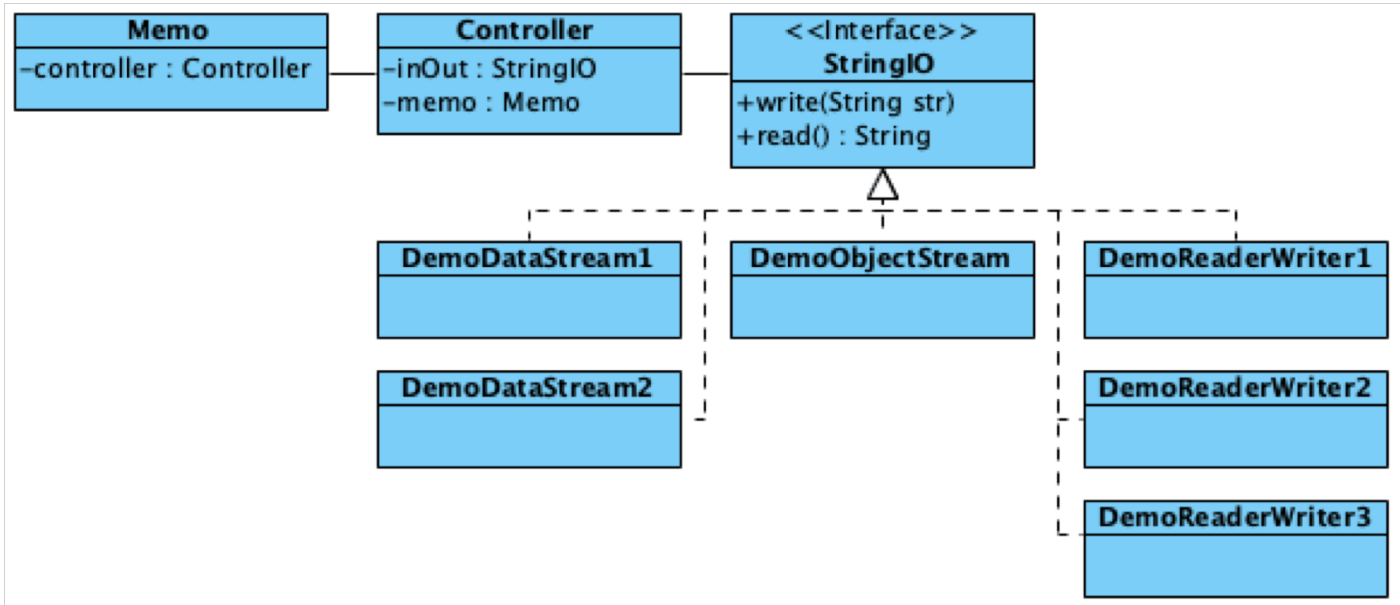
Design i kommande exempel

Interfacet ***StringIO*** implementeras av ***Demo…***-klasserna:

```
public interface StringIO {
   public void write(String s);
   public String read();
}
```

Controller har en implementering av ***StringIO*** vilken används för att:

- Skriva data till fil på hårddisken
- Hämta data från fil på hårddisken



DataOutput / DataInput

DemoDataStreams1 ger exempel på hur man skriver/läser primitiva datatyper och strängar med DataOutputStream/DataInputStream.

Det gäller att läsa från filen i samma ordning som man skrivit till filen.

Skriver till filen

```
dos.writeInt(str.length());
dos.writeBoolean(str.length()>20);
dos.writeUTF(str);
dos.flush();
```

Läser från filen

```
int len = dis.readInt();
boolean longStr = dis.readBoolean();
String txt = dis.readUTF();
```

DemoDataStreams1

DataOutput / DataInput

DemoDataStreams2 ger exempel på hur samma data kan krypteras vid skrivning till fil och dekrypteras vid läsning från fil.

```
// Det behövs ett chiffer för kryptering vid skrivningen
Cipher cipher = getCipher(Cipher.ENCRYPT_MODE, "DA343ACipherTest");
CipherOutputStream cos = new CipherOutputStream(new BufferedOutputStream(
      new FileOutputStream(fileChooser.getSelectedFile().getPath())), cipher);
DataOutputStream dos = new DataOutputStream(cos)

// Det behövs ett chiffer för dekryptering vid läsningen
Cipher cipher = getCipher(Cipher.DECRYPT_MODE, "DA343ACipherTest");
CipherInputStream cis = new CipherInputStream(new BufferedInputStream(
       new FileInputStream(fileChooser.getSelectedFile().getPath())), cipher);
DataInputStream dis = new DataInputStream(cis);
```

Metoden ***getCipher*** returnerar ett ***Cipher***-objekt.

```
private Cipher getCipher(int opmode, String secretKey) {
   Cipher cipher = null;
   try {
      Key key = new SecretKeySpec(secretKey.getBytes("UTF-8"),"AES");
      cipher = Cipher.getInstance("AES");
      cipher.init(opmode, key);
   } catch(Exception e) {…}
   return cipher;
}
```

DemoDataStreams2

Skriva objekt

Man skapar en ström till en fil för att skriva:

```
FileOutputStream fos = new FileOutputStream("C:/temp/ex.dat");
```

För att skriva objekt så kopplar man strömmen till ett objekt av typen ObjectOutputStream:

```
ObjectOutputStream oos = new ObjectOutputStream(fos);
```

Skrivning sker med metoden

```
public void writeObject(Object obj)
```

```
public void write(String str) {
   String part1 = str.substring(0, str.length()/2);
   String part2 = str.substring(str.length()/2);
   if (fileChooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
     try( FileOutputStream fos = new FileOutputStream(fileChooser.getSelectedFile().getPath());
          BufferedOutputStream bos = new BufferedOutputStream(fos);
          ObjectOutputStream oos = new ObjectOutputStream(bos) ) {
        oos.writeInt(2); // DataOutput implementeras
        oos.writeObject(part1); // Ett String-objekt skrivs
        oos.writeObject(part2); // Ett String-objekt skrivs
        oos.flush();
     } catch(IOException e) {
        System.err.println(e);
     }
  }
}
```

DemoObjectStreams

Objektet som skrivs måste vara en instans av en klass som implementerar gränssnittet **Serializable** eller **Externalizable**.

Läsa objekt

Man skapar en ström till en fil för att läsa:

```
FileInputStream fis = new FileInputStream(”C:/temp/ex.dat”);
```

För att läsa objekt så kopplar man strömmen till ett objekt av typen **ObjectInputStream**:

```
ObjectInputStream ois = new ObjectInputStream(fis);
```

Läsning sker med metoden

```
public Object readObject()
```

vilken returnerar en referens till ett objekt vilket måste typkonverteras till korrekt referens.

```
public String read() {
   String str = null;
   if (fileChooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
      try (ObjectInputStream ois = new ObjectInputStream( new BufferedInputStream(
            new FileInputStream(fileChooser.getSelectedFile().getPath())))) {
         int len = ois.readInt(); // DataInput implementeras
         str = len+"\n"+(String)ois.readObject()+"\n"+(String)ois.readObject();
      } catch(IOException e) {
         System.err.println(e);
      } catch(ClassNotFoundException e2) {
         System.err.println(e2);
      }
   }
   return str;
}
```

DemoObjectStreams

Objektet som läses måste vara en instans av en klass som implementerar gränssnittet **Serializable** eller **Externalizable**.

Serializable

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);
    }
}
```

Läsa Person-array

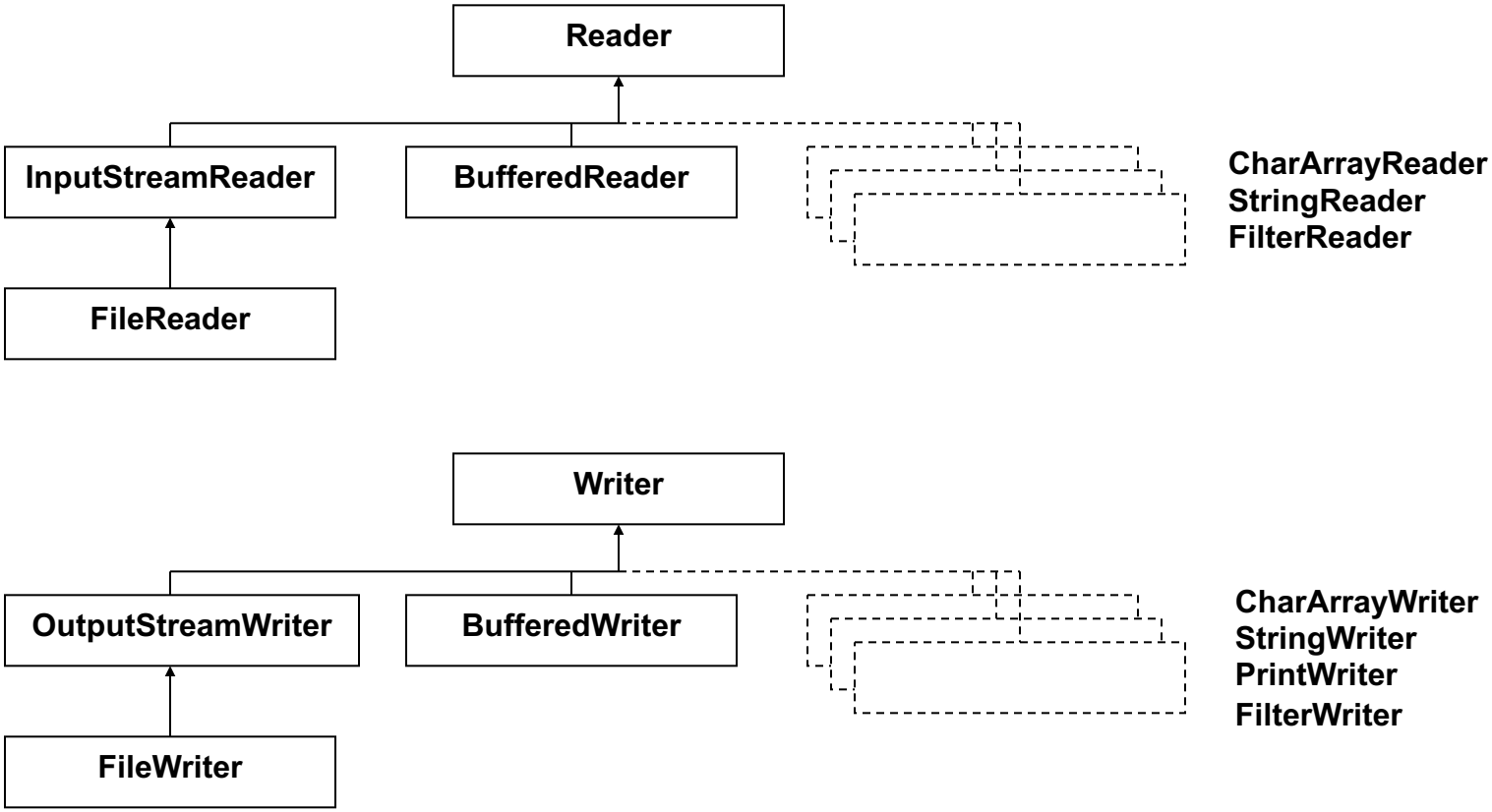
```
private static Person[] readPersons() throws IOException {
    Person[] pers=null;
    try(ObjectInputStream ois = new ObjectInputStream(
                       new FileInputStream("files/personer.dat"))) {
       pers = (Person[])ois.readObject();
    }
    catch(ClassNotFoundException e) {}
  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

Person.java

PersonTest.java

Textströmmar

Grundläggande klasser för att läsa och skriva text är **Reader** och **Writer**. Båda klasserna är abstrakta.



Textströmmar, att skriva text

Writer-klasser ärver 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)	Skriver **cbuf** till strömmen.
public abstract void write(char[] c, int start, int len)	Skriver **c** till strömmen, från position **start** och **len** tecken.
public void write(String s)	Skriver **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***-klasser 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

Samtliga ovanstående metoder kan kasta undantag av typen **IOException**.

Textströmmar, att läsa text

Reader-klasser ärver klassen ***Reader***, vilken innehåller ett antal metoder, bl.a.

public void close()	Stänger strömmen.
public int read()	Läser ett tecken. Returnerar -1 om strömmen ej 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***-klasser 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.

Samtliga ovanstående metoder kan kasta undantag av typen **IOException**.

Skriva text till fil på hårddisk

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

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

FileWriter-objektet använder default teckenkodning vid skrivningen.

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

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

Skrivning ske ofta med metoderna

```
write(String) + flush()
```

```
public void write(String str) {
   :
   BufferedWriter bw = null;
   try {
      String filename = …;
      bw = new BufferedWriter(new FileWriter(filename));
      bw.write( str );
      bw.flush();
   } catch(IOException e) {
      System.err.println(e);
   } finally {
      if(bw != null) {
         try {
            bw.close();
         } catch(IOException e){}
      }
   }
}
```

DemoReaderWriter1

Läsa text från fil på hårddisk

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

```
FileReader fr = new FileReader(”temp/ex.txt”);
```

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

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

Läsning kan ske med metoden

- **read**()

vilken returnerar värden av typen int. **–1** innebär att slutet på filen har nåtts.

```
public String read() {
  StringBuffer str = new StringBuffer();
  int chr;
  :
  try( BufferedReader br = new BufferedReader(new FileReader(filnamn))) {
    chr = br.read();                          // läs en int
    while(chr != -1) {                        // så länge filen inte är slut
      str.append((char)chr);                  // typkonvertera till char och gör
      chr = br.read();                        // något med den
    }
  } catch(IOException e) {
    System.err.println( e );
  }
  return str.toString();
}
```

DemoReaderWriter1

Skriva text till fil på hårddisk

Om man önskar ange teckenkodning för textfilen använder man ***FileOutputStream*** tillsammans med ***OutputStreamWriter***:

```
FileOutputStream fos = new FileOutputStream(”temp/ex.txt”);
OutputStreamWriter osw = new OutputStreamWriter( fos, "ISO-8859-1" );
```

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

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

```
public void write(String str) {
   :
   BufferedWriter bw = null;
   try {
      String filename = …;
      bw = new BufferedWriter( new OutputStreamWriter(
                  new FileOutputStream(filename),”ISO-8859-1” ) );
      bw.write( str );
      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

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

```
FileInputStream fis = new FileInputStream(”temp/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 );
```

```
public String read() {
  StringBuffer str = new StringBuffer();
  int chr;
  :
  try( BufferedReader br = new InputStreamReader(
                new FileInputStream(filnamn), "ISO-8859-1" ) ) {
    chr = br.read();                          // läs en int
    while(chr != -1) {                        // så länge filen inte är slut
      str.append((char)chr);                  // typkonvertera till char och gör
      chr = br.read();                        // något med den
    }
  } catch(IOException e) {
    System.err.println( e );
  }
  return str.toString();
}
```

DemoReaderWriter2

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 ske ofta med metoderna

```
write(String) + newLine() + flush()
```

```
public void write(String str) {
  String[] rows;
  :
  rows = str.split(“\n”);
  for(String s : rows) {
    out.write(s);          // skriv rad
    out.newLine();         // Skriv radslut
    out.flush();           // För over till hårddisken
  }
  :
}
```

DemoReaderWriter3

Läsa text från fil på hårddisk

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

```
FileInputStream fis = new FileInputStream(”temp/ex.txt");
InputStreamReader inr = new InputStreamReader ( fis, "UTF-8" );
```

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

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

Läsning kan ske med metoden

- **readLine()**

vilken returnerar en String. **null** innebär att slutet på filen har nåtts.

```
public void read() {
   StringBuffer res = new StringBuffer();
   String str;
   :
   try (BufferedReader br = new BufferedReader(new InputStreamReader(
            new FileInputStream( filename), "UTF-8" )) {
      str = br.readLine();
      while ( str!=null ) {
         res.append(str+"\n");
         str = br.readLine();
      }
   } catch(IOException e) {
      System.err.println(e);
   }
   return res.toString();
}
```

DemoReaderWriter3