Dataströmmar - streams

- Kan ses som en kommunikationsväg för data från en källa till en annan
- Data som flödar in från en källa kallas för indataström (inputstream)
- Data som flödar ut till en källa kallas för utdataström (outputstream)
- Algoritmen för in och ut -dataströmmar är i grund den samma:

Indataström:

1. Öppna en indataström
2. Så länge det finns data att läsa - läs den från indataströmmen
3. Stäng indataströmmen

Utdataström:

1. Öppna en utdataström
2. Så länge det finns data att skriva - skriv den till utdataströmmen
3. Stäng utdataströmmen

Java I/O

- I paketet java.io finns klasser för att hantera in och ut -dataströmmar i java
- Klasserna indelas i två hierarkier: **byte-streams** och **charachter-streams**
- **Byte-streams:** överför data i bytes (grupper om 8-bitar)
- **Charachter-streams:** överför data i 16 bitars Unicode-format
- Klasser som har "Stream" i namnet är klasser som hanterar byte-stremas
- Klasser som har "Reader" eller "Writer" i namnet är klasser som hanterar charachter-streams

Klasserna `Reader` och `Writer`

- Abstrakta superklasser för att läsa från och skriva till Character streams
- Skriver och läser 16 bitars tecken
- Subklasser till dessa används när text skall läsas eller skrivas
- Dessa klasser är exempel på low-level readers och writers

Klasserna `InputStream` och `OutputStream`

- Abstrakta superklasser för att läsa och skriva byte streams
- Skriver och läser i bytes om 8 bitar
- Byte streams används för att läsa och skriva binärdata som t.ex. bilder, ljud och video objekt-serialisering
- Dessa klasser är exempel på Low-level streams

Metoder i de abstrakta superklasserna - likheter

- `Reader` och `InputStream` har ett liknande API men för olika datatyper:

```
// Reader
int read()
int read(char cbuf[])
int read(char cbuf[], int offset, int length)

// InputStream
int read()
int read(byte buf[])
int read(byte buf[], int offset, int length)
```

- `Writer` och `OutputStream` har också ett likande API men för olika datatyper:

```
// Writer
int write(int c)
int write(char cbuf[])
int write(char cbuf[], int offset, int length)

// OutputStream
int write(int c)
int write(byte buf[])
int write(byte buf[], int offset, int length)
```

Andra Low-level Streams

- `FileInputStream`: Läser bytes från en fil

```
FileInputStream input = new FileInputStream("input.file");
int c = 0;
while ((c = input.read()) != -1) {
// Gör något med aktuell byte
}
input.close();
```

- `FileOutputStream`: Skriver bytes till en fil

```
FileOutputStream output = new FileOutputStream("output.file");
byte[] bytes = // någon form av byte[]
output.write(bytes);
output.close();
```

Andra Low-level readers och writers

- `FileReader`: Läser chars från en fil

```
FileReader reader = new FileReader("document.txt");
int c = 0;
while ((c = reader.read()) != -1) {
      System.out.print((char)c);
}
```

- `FileWriter`: Skriver chars till en fil

```
FileWriter writer = new FileWriter("document.txt");
char[] chars = "Text att skriva".toCharArray();
writer.write(chars);
writer.close();
```

- Notera:
  `FileReader` och `FileWriter` skriver 16-bitars tecken (UniCode) medan många "native file systems" använder bytes (8-bitar). För att detta skall vara möjligt gör dessa streams omkodningar när de skriver och läser. Denna omkodningen baseras på default character-encoding för plattformen. För att få reda på vilken encoding som används av plattformen kan man anropa `System.getProperty("file.encoding");`

High-level readers och writers

- Arbetar mot Low-level readers och writers istället för I/O-enheter som filer eller sockets
- `BufferedReader`: Läser text från en low-level character-input stream samt buffrar texten
- Fördelen med Buffered-streams är att de ger prestandahöjning

```
BufferedReader reader = new BufferedReader(new FileReader("document.txt"));
String line = null;
while ((line = reader.readLine()) != null) {
      System.out.println(line);
}
reader.close();
```

- `BufferedWriter`: Buffrar text som den sedan skriver till en low-level character-output stream

```
BufferedWriter writer = new BufferedWriter(new FileWriter("document.txt"));
writer.write("Writing some text...");
writer.close();
```

- Notera:
  När buffer är full så kommer den automatiskt att skrivas till underliggande stream. Samma sak gäller när `close()` anropas. För att skriva det som ligger i buffer kan `flush()` anropas.

High-level streams

- Arbetar mot Low-level streams istället för I/O-enheter som filer eller sockets
- `DataInputStream`: Läser primitiver från en input-stream

```
DataInputStream input = new DataInputStream(new FileInputStream("data.dat"));
double d = input.readDouble();
int i = input.readInt();
input.close();
```

- `DataOutputStream` : Skriver primitiver till en output-stream

```
DataOutputStream output = new DataOutputStream(new FileOutputStream("data.dat"));
output.writeDouble(100.23);
output.writeInt(3445);
output.close();
```

- Andra användbara high-level streams:
  `BufferedInputStream`: stream som har en buffer med inlästa bytes
  `BufferedOutputStream`: stream som har en buffer för bytes som skall skrivas

Att tänka på

- När man arbetar med Streams skall man alltid se till att stänga dessa när man är klar
- `Reader`, `Writer`, `InputStream` och `OutputStream` implementerar alla `Cloaseable`
- Använd Buffered-streams/writers för att öka prestandan

Exempel:

// Utan buffer

```
DataOutputStream out = new DataOutputStream(new FileOutputStream("output.dat"));
```

//Med buffer

```
DataOutputStream out = new DataOutputStream(new BufferedOutputStream(
                                            new FileOutputStream("output.dat")));
```

Klassen `File`

- Representerar en fil eller en mapp som kan existera i filsystemet - innehåller inte filen/mappen
- Innehåller bland annat metoderna:

  `isFile()` - undersöker om File-objektet representerar en fil

  `createNewFile()` - skapar en ny fil

  `mkdir()` - skapar en ny mapp

  `delete()` - tar bort filen eller mappen

  `getName()` - returnerar namnet på filen eller mappen

  `exists()` - undersöker om en filen eller mappen existerar

  `list()` - returnerar en `String[]` med namnen på de filer och mappar i mappen som File-objektet representerar

- File-objekt kan används som argument till många av stream-klassernas konstruktorer

```
File file = new File("document.txt");
BufferedWriter writer = new BufferedWriter(new FileWriter(file));
```

Övning

- Övningen går ut på att skriva ner all information om en eller en samling av Users till en textfil
- Funktionskrav:
  - Det skall finnas en metod som tar emot en List<User> och skriver ner dessa till en fil med namnet: **`users.txt`**
  - Det skall finnas en metod som tar emot en User och skriver ner informationen till en fil som har namnet: **`[userid][username].txt`** (ex. 1001anca01.txt)
  - Om någon av filerna redan existerar skall dessa skrivas över
  - Båda metoderna skall skriva ner filerna i mappen **users** som skapas om den inte redan existerar
  - När ditt program startas och det redan finns sparade användare (antigen i users.txt eller de enskilda filerna) läsas in och göras om till objekt som lagras i minnet
  - Användare som läses tillbaka från fil ska gå att skriva ut