Arne Maus, ver.2.0, september, 2012

Bruk av class Scanner, FileWriter og Formatter som alternativ til EasyIO i INF1000.

Noen ønsker å bruke noen av de klassene vi finner i pakken 'java.util' og 'java.io' isteden for easyIO i INF1000. Kurset og læreboka "Rett på Java" beskriver på s. 66-71 bruk av klassene Scanner til innlesning PrintWriter til utskrift og Formatter til formattering av særlig tall til senere utskrift. Dette notatet prøver på en enkel måte å gi noen tips til bruk av disse klassene, og dekker enkel innlesning og utskrift av tall og tekster.

Vi presiserer at det er helt greit å bruke å bruke disse klassene på obligatoriske oppgaver og til eksamen, men at vi anbefaler easyIO fordi vi mener den pakken (som også er implementert ved bruk av standard Java klasser), er enklere for studentene.

Innlesning

Innlesning kan skje både fra fil og fra tastaturet, og det er grovt sett tre måter å lese f.eks en fil på:

1. tegn for tegn (alt leses)
2. feltvis (f.eks tall etter tall hvor vi hopper over blanke , linjeskift o.l)
3. linjevis.

EasyIO har metoder for alle de tre måtene å lese på og de kan blandes under lesing av en fil. Klassen Scanner har innlesningsmuligheter for de to siste måtene. Når vi skal lese fra fil eller tastatur må vi først opprette et leseobjekt. Det kobler ditt program sammen med den filen eller tastaturet vi leser fra. Så kaller vi lesemetoder i dette objektet for å foreta selve lesingen . Til slutt bør vi lukke vi leseobjektet.

Opprettelse av et leserobjekt:

For å lese fra tastaturet

easyIO	Scanner

import easyIO.*; // må installeres
In tast = new In():
	import java.util.*; // er installert
	Scanner sc = new Scanner (System.in);

For å lese fra en fil

Både easyIO og Scanner opererer med en innlesningsmarkør som forteller hvor langt vi har kommet i lesingen av fila. Vi leser så fra starten av filen og utover mot slutten av fila.

Når det leses med Scanner fra en fil må vi legge åpningen (og all senere lesing) inn i en try-catch blokk fordi man skal kunne fange opp mulige feil (som at en fil med det navnet ikke finnes).

easyIO	Scanner

import easyIO.*;
In f = new In("Data1.txt"):
	import java.io.*; // for klassen: File
	import java.util.*; // for klassen Scanner
	Scanner f ;
	try{
	f = new Scanner(new File("Data1.txt"));
	} catch (Exception e) { /* legg feilmelding her */ f = null; }

Grunnen til at f må deklareres utenfor try-catch blokken, er at man ikke er sikker på at try-grenen blir utført, og da har man ikke laget noe leser-objekt f. Tilordningen f = null er også viktig i catch-grenen da resten av koden nå vet at f har en gyldig verdi uansett om try- eller catch-grenen ble utført (null er en gyldig verdi, men den kan jo senere gi problemer). Alternativt kunne f fått 'null'verdien i deklarasjoner (Scanner f=null ;).

Lese en linje fra en tastaturet

Når det leses med Scanner fra en fil er det veldig likt easyIO, med det unntaket at easyIO har en metode ' inLine ()' som leser neste ikke-tomme linje:

easyIO	Scanner

String l = tast.readLine() ; // leser resten av
// inneværende linje
 	
String l = tast.inLine() ; // leser resten av inneværende
// linje og hvis den er tom, neste ikke-tomme linje
	String l = sc.nextLine(); // leser resten av
	// inneværende linje

Lese en linje fra en fil

Når det leses med Scanner fra en fil må vi legge åpningen (og all senere lesing) inn i en try-catch blokk fordi man skal kunne fange opp mulige feil (som at en fil med det navnet ikke finnes).

easyIO	Scanner

String t = f.readLine() ; // leser resten av
// inneværende linje
 	
String t = f.inLine() ; // leser resten av inneværende
// linje og hvis den er tom, neste ikke-tomme linje
	String t = null;
	try{
	t= f.nextLine();// leser resten av
	// inneværende linje
	} catch (Exception e) { }

Lese et ord (ikke hele linja) fra en fil og fra tastatur

Når vi leser et ord (adskilt foran og bak med det som vi betrakter som skilletegn, for eksempel blanke og linjeskift, er det omlag samme kode fra fil eller tastatur. Igjen må lesingen med Scanner legges i en try-catch blokk.

a) fra tastatur:

easyIO	Scanner

String tast = tast.next() ; // leser neste ord fra tastatur
// hopper over alle tomme linjer
	String t = sc.next(); // leser neste ord fra tastatur
	// hopper over alle tomme linjer

b) fra fil:

easyIO	Scanner

String t = f.next() ; // leser neste ord fra fila	String t = null;
	try{
	t = f.next(); // leser neste ord fra fila
	} catch (Exception e) { }

Her må Scanner deklarere Stingen t utenfor try-catch blokken for å gjøre dem synlige i resten av programmet. I tillegg må man enten i catch-blokka eller ved deklarasjonen gi en gyldig verdi til t – ellers protesterer oversetteren 'javac'.

Lese et heltall og et flyttall fra en fil og fra tastatur

Når vi leser et tall, adskilt foran og bak med det som vi betrakter som skilletegn, f.eks blanke og linjeskift, er det om lag samme kode fra fil eller tastatur. Igjen må lesingen med Scanner legges i en try-catch blokk når vi leser en fil.

a) fra tastatur:

easyIO	Scanner

int i = tast.nextInt() ; // leser neste heltall fra tastatur
double x = tast.nextDouble();//leser flyttall
	int i = sc.nextInt(); // leser neste heltall fra tastatur
	double x= sc. nextDouble(); // leser flyttall

b) fra fil:

easyIO	Scanner

int i = f.nextInt() ; // leser neste heltall fra tastatur
double x = f.nextDouble();//leser flyttall
	double x = 0;
	int i =0;
	try{
	i = f.nextInt(); // leser neste heltall fra fil
	x= f. nextDouble(); // leser flyttall
	} catch (Exception e) { }

Igjen må innlesning med Scanner deklarere variablene x og i utenfor try-cath blokken for å gjøre dem synlige i resten av programmet. I tillegg må man enten i catch-blokka eller ved deklarasjonen gi en gyldig verdi til x og i – ellers protesterer kompilatoren.

I tillegg har både easyIO og Scanner en rekke metoder (hasNext(), hasNextInt(), nextDouble(),...) som for hver lesemetode sjekker om det neste som skal leses er et ord, et heltall, et flyttall osv og returnerer sant (true) hvis det vi tester er tilfellet – elles usant(false).

Utskrift

Utskrift er enklere enn innlesning fordi det er færre feilsituasjoner.

I java har man flere muligheter istedenfor easyIO-klassen Out. Rett på Java gir god oppskrift om utskrift på skjerm med klassen Out, men for skriving til skjermen man kan like gjerne bruke System.out.println() og System.out.print(). For utskrift til fil har man f.eks klassene PrintWriter og FileWriter, hvor det synes enklest å bruke FileWriter.

Klassen Out i easyIO hvor det er en rekke metoder for å skrive ut de ulike datatypene (String, char, int double) alt etter hvordan man vil ha utskriften – antall tegnplasser man vil ha på linje, antall plasser etter komma, høyre-eller venstre-juster osv. Nedenfor vises bare noen få eksempler. Klassen FileWriter bruker et annet prinsipp, den skriver ut bare en tekst (akkurat som System.out.println()). Det er så opp til brukeren å formattere bredde, antall sifre etter komma osv. før tallet omgjøres til tekst og så skrives ut. Husk at alle skriveobjekter som skriver til fil ***må*** lukkes til slutt for at alt man har skrevet skal skrives ut på fila.

Lage et skriverobjekt til fil

easyIO	import easyIO.* Out fil1 = newOut("FilenMin.txt"); // Åpner fil med skriving fra starten

FileWriter	import java.io.*;
	FileWriter f0 = null;
	try{
	f0 = new FileWriter("file1.txt"); // Åpner fil med skriving fra starten
	} catch (Exception e) { }

easyIO	import easyIO.* Out fil2 = newOut("FilenMin2.txt", true); // Åpner fil med skriving fra enden

FileWriter	import java.io.*;
	FileWriter f= null;
	try{
	f = new FileWriter("file2.txt", true); // Åpner fil med skriving fra enden
	} catch (Exception e) { }

Utskrift på fil med easyIO

Alle metodene i klasse Out heter enten:

```
out – skriver ut et tall, tekst eller bokstav uten linjeskift etter
```

, eller:

```
outln – skriver ut et tall, tekst eller bokstav med linjeskift etter
```

Disse metodene har ulike antall parametre som bestemmer særlig bredden på det feltet man skriver ut, justeringen (RIGHT, CENTER eller LEFT) av det man skriver ut eller evt. antall sifre etter komma for flyttall. Velger man ikke, vil tall bli høyrejustert og tekster venstrejustert og velger man ikke bredde vil det man skriver ut få så mange plassert på linja som de trenger for å bli skrevet ut, eks:

```
// skriver ut et tall (123) på 3 plasser
 fil1.out(123);

// skriver ut et tall (123) høyrejustert på 5 plasser
 fil1.out(123,5);

 // skriver ut linjeskift
 fil1.outln();

 // utskrift med linjeskift.
 fil1.outln("En linje med tekst");

 // skriver ut desimaltallet 123.456 med to desimaler med feltbredde 6
 fil1.outln(123.456, 2);

 // som over, høyrejustert på 10 plasser.
 fil1.outln(123.456, 2, 10);

 // Teksten skrives høyrejustert på 10 plasser.
 fil1.outln("Til høyre", 10, OutExp.RIGHT);

 // Lukker filen etter skriving.
 fil1.close();
```

Utskrift på fil med FileWriter

Som sagt skriver FileWriter 'bare' ut tekster (som System,out.println()), og hvis man vil formatterer tekst eller særlig tall før utskrift, må man også bruke klassen Formatter fra 'java.io'. Man kan da med Formatter fortsette å skrive inn i en fil man allerede har åpnet med FileWriter:

```
try{   FileWriter f1 = new FileWriter("file1.txt");

       f1.write("Her er svaret:");

       double x = 2,1734;

       Formatter fm = new Formatter(f1);  // for å skrive videre på f1

       fm.format("%12d %s %4.3f %6.2f",9876543,"\n Hurra", 77.6666, x);

       f1.write("SLUTT");

       f1.close();

} catch(Exception e) { }
```

Vi har her et eksempel på at vi først lager en fil "file1.txt" med FileWriter og skriver teksten "Her er svaret:" på fila. Så åpner vi en Formatter på den fila og skriver ut ett tall, en tekst et tall og tilslutt en variabel som har et tall som verdi og som da skrives ut.

Det som blir skrevet på fila er:

```
Her er svaret:     9876543

 Hurra 77,667  2,17

SLUTT
```

For å formattert et tall til så må man i format først angi en tekst (`"%12d %s %4.3f %6.2f"`) som sier hvordan resten av data og variable etter denne stringen skal skrives.

Her er en særs enkel forklaring på hvordan du kan skrive selve formatterings-stringen (den fulle bruksanvisningen for dette er på mange sider): Vi ser at den står for adskilte deler som hver begynner med %, og det er like mange slike %-deler som det er tall, tekster eller variable etter denne (`9876543,"\n Hurra", 77.6666, x`). Betydningen av de like delene i formatteringsstringen er:

%12d – d betyr heltall og 12 sier at feltet som skal nyttes her er 12 langt

%s - s sier at her skal det være en string(tekst)og at den tar så stor plass som den er lang

%4.3f – f betyr flyttall, og at det skal skrives med 4 plasser før og 3 sifre etter komma

%6.2f – som ovenfor, men med 6 sifre før og 2 sifre etter komma.

Legg merke til at det ikke bare er formatterings-stringen som formatterer. Teksten `"\n Hurra"` blir skrevet ut som først linjeskift, så en blank og så Hurra. "\n" inne i tekster har alltid denne effekten.