Ukeoppgaver 4: 14. - 18. sep (INF1000 - Høst 2009)

*Teori for Oblig 2: Mer om **løkker og arrayer** (kap. 4 - 5); og **metoder** (kap. 7.1 - 7.9)*

Mål
Få øvelse i teorien som trengs for å løse **Oblig 2**. Hovedtema i obligen er løkker, arrayer, og metoder (uten parametre), og følgende oppgaver gir nyttig trening i dette slik at du blir bedre i stand til å løse obligen.

Oppgaver til teoritimen

1. **Enkle løkker med tall:**
   (a) Lag et program med en løkke som teller ned fra 10 til 0. Bruk en teller-variabel med startverdi 10 og redusér den med 1 i hver gjennomgang av løkka. Utskriften skal bli:

```
...10 ...9 ...8 ...7 ...6 ...5 ...4 ...3 ...2 ...1 ...0
```

   (b) Lag en løkke som skriver ut de 10 første potensene av 2 (2, 2×2, 2×2×2, osv). Bruk en teller-variabel "`i`" som teller de 10 gangene, og en variabel "`potens2`" som ganges med 2 i hver omgang av løkka og skrives ut. Utskriften skal bli:

```
2  4  8  16  32  64  128  256  512  1024
```

   (c) Lag to nestede for-løkker som gir følgende utskrift. Det skal bare skrives ut ett tall av gangen, som skal være verdien til telleren i den ytre løkka (som går fra 1 til 3) ganget med telleren i den indre løkka.

```
1  2  3  4
2  4  6  8
3  6  9  12
```

2. **Løkker: Hva blir skrevet ut?**
   Avgjør uten å bruke datamaskin hva som blir skrevet ut når følgende programsetninger utføres.

```
//(a)
       int a = 10;
       while (a < 20) {
           a += 4;
       }
       System.out.println("a = " + a);

//(b)
       int sum = 0;
       for (int b = 1; b < 6; b += 2) {
           sum += b;
       }
       System.out.println("sum = " + sum);

//(c)
       int produkt = 1;
       for (int c = 1; c < 4; c++) {
           produkt = produkt * c;
           System.out.println(produkt);
       }

//(d)
       for (int d = 3; d >= 1; d--) {
           for (int e = 1; e <= 3; e++) {
               System.out.println(d + e);
           }
       }

//(e)
       int teller = 0;
       for (int ytre = 0; ytre < 3; ytre++) {
           teller++;
           for (int indre = 0; indre < 3; indre++) {
               teller++;
           }
       }
       System.out.println(teller);
```

3. **Arrayer med tall:** *(Oblig-relevant!)*
(a) Lag en løkke som setter inn de 10 første oddetall i følgende array. La telleren i løkka gå fra 0 til 9, og i hver gjennomgang av løkka setter du inn én verdi i arrayen, beregnet som telleren ganget med 2 pluss 1.

```
int[] oddetall = new int[10];
```

(b) Produkt av verdier != 0: Anta at arrayen `verdier[]` og variabelen `produkt` er deklarert som vist under. Lag en løkke som går gjennom verdiene i arrayen, og hver gang den kommer til en verdi som ikke er 0 ganger du verdien med `produkt` og lagrer resultatet i samme variabel (`produkt`). Utskriften til slutt skal være `240` (dvs. 2×4×10×3).

```
int[] verdier = { 0, 2, 4, 0, 0, 10, 0, 3 };
int produkt = 1;
// Løkke:
for (                                    ) {

}
System.out.println(produkt);
```

(c) 2D-array: Lag to nestede for-løkker som setter inn følgende verdier i en 2D-array deklarert som: `int[][] tabell = new int[3][4];`, slik at `tabell[0][0]` blir 1, `tabell[2][3]` blir 12, osv. Husk at indeksene i arrayen starter fra 0, ikke 1.

```
1  2  3  4
2  4  6  8
3  6  9  12
```

(d) Sum i kolonner: Skriv programkode som beregner summen av verdiene i hver kolonne i ovennevnte array og skriver summene ut (`6 12 18 24`). *Dette ligner på en deloppgave i oblig 2!*

4. **Array med String-er: Hva blir skrevet ut?** *Og hvorfor?* *(Oblig-relevant!)*

```
class NavneArray {
   public static void main(String[] args) {

       String[] navn = { "Anne", "Kari", "Ole", null };

//(a)
       System.out.println(navn[1] + navn[navn.length/2]);

//(b)
       for (int i = 0; i < navn.length; i++) {
           // Testen "!= null" sikrer at neste ledd ikke blir null.equals(..)
           if (navn[i] != null
               && (navn[i].equals("Ole") || navn[i].equals("Anne")) ) {
               System.out.println(i);
           }
       }

//(c)
       int indeks = 0;
       boolean funnet = false;
       while (indeks < 4 && !funnet) {
           if (navn[indeks].equals("Kari")) {
               funnet = true;
           }
           indeks++;
       }
       System.out.println(indeks);

//(d)
       String[] andreNavn = { "Per", "Anne", "Ole" };
       for (int i = 0; i < 4; i++) {
           for (int j = 0; j < 3; j++) {
               // Testen "!= null" sikrer at andre ledd ikke blir null.equals(..)
               if (navn[i] != null && navn[i].equals(andreNavn[j])) {
                   System.out.println(i + " " + j);
               }
           }
       }
   }
}
```

5. **inLine() og inChar():** *(Oblig-relevant!)*
Lag et program som spør bruker etter et navn. Programmet leser navnet inn vha. `inLine();`, og "fyller" så skjermen med

navnet ved å skrive det ut 100 ganger. Videre skal programmet spørre bruker om hun vil gi et nytt navn (j/n). Svaret leses nå med .inChar("\n\r"), og hvis det er 'j' gjentas hele prosessen; hvis svaret er 'n' avsluttes programmet. Kjøreeksempel:

```
Skriv et navn: Ola Nordmann
Ola Nordmann Ola Nordmann Ola Nordmann Ola Nordmann Ola Nordmann O
la Nordmann Ola Nordmann Ola Nordmann Ola Nordmann Ola Nordmann Ol
a Nordmann Ola Nordmann Ola Nordmann Ola Nordmann Ola  ...osv...
Gi nytt navn? (j/n): j
Skriv et navn:
```

Tips til Oblig 2: Legg merke til det som står i parentesene til .inChar("\n\r") ovenfor. Dette angir at inChar skal hoppe over evt. linjeskift (\n) eller vognretur (\r) som bruker har tastet inn, og i stedet lese inn en vanlig bokstav fra tastatur, f.eks. 'j'. Det samme bør du gjøre i deloppgave 1 *(c)* i oblig 2.

6. **Metoder:** *(basert på forelesningen uke 4 (PDF), lysark side 28)* *(Oblig-relevant!)*
(a) Endre strukturen til programmet ditt fra punkt 5. over slik at det følger malen fra oblig 2 (vist i skissen nedenfor), dvs. med en liten kontrollklasse øverst, etterfulgt av en egen klasse for metodene. Hele programmet med begge klassene lagres i én fil, kalt Navn100.java (dvs. navnet til klassen med metoden main()). Lag bare én metode i hjelpeklassen, kalt ordreløkke(), som gjør alt som står i punkt 5. ovenfor.

```
import easyIO.*;

class Navn100 {
    public static void main(String[] args) {
        Hjelpeklasse hj = new Hjelpeklasse();
        hj.ordreløkke(); // Kjører metoden ordreløkke() i hjelpeklassen
    }
}

class Hjelpeklasse {
    // Klargjøring for innlesing/utskrift, gjelder for hele klassen:
    In tast = new In();
    Out skjerm = new Out();

    String navn;

    void ordreløkke() {
        char giNyttNavn = 'j'; // Startverdi

        while (giNyttNavn != 'n') {
            // - Be bruker taste et navn og les det inn med .inLine();
            // - Utskrift av navn 100 ganger.
            // - Spør om bruker vil "Gi nytt navn? (j/n):", og .inChar("\n\r"):


        }
    }
}
```

Mer info: Grunnen til at vi skriver programmet på denne måten med to klasser vil bli klarere når vi kommer til kapittel 8, men har sammenheng med at vi ønsker å lage gode "objektorienterte" program der vi jobber med "objekter". I denne skissen er det fem pekere til objekter: args[], hj, tast, skjerm, og navn.

(b) Flere metoder: Endre programmet slik at koden som skriver ut navnet 100 ganger flyttes til en egen metode kalt utskrift(). Husk å legge inn et kall på metoden på det stedet i programmet du flyttet koden fra.

(c) Inn-parameter: Endre programmet slik at String-variabelen navn nå deklareres inne i metoden ordreløkke() (og ikke "globalt" - dvs. før alle metodene, som vist ovenfor), og slik at verdien av denne variabelen blir overført til metoden utskrift(..) vha. et argument i kallet. Endre også begynnelsen av metoden utskrift(..) slik at den tar imot argumentet vha. parameteren String navn, slik: void utskrift(String navn) {

7. **Metode med inn og ut-parametre:** *kap. 7, oppg. 2 (side 133)*
Lag en metode som regner ut hypotenusen *c* i en rettvinklet trekant når vi går ut fra Pytagoras formel: $c^2 = a^2 + b^2$ der *a* og *b* er lengden på katetene - de to andre sidene i trekanten. Vi bruker *a* og *b* som parametre til metoden. Du trenger da å kalle kvadratrotmetoden i Math.sqrt(*double_verdi*) i den metoden du lager. Returner verdien *c* som verdien på metoden. Test metoden ved å kalle den i en dobbel for-løkke for alle kombinasjoner av *a* og *b* med heltallsverdiene fra 1.0 til og med 6.0, og skriv ut svarene.

```
double finnHypotenus(double a, double b) {
    ...

    return c;
}
```

Tips: Kallet på metoden kan se slik ut: double c = finnHypotenus(a, b);

8. **Metode med array som inn-parameter:** *kap. 7, oppg. 3 (side 134)*

Lag en metode `double gjennomsnitt(int[] a)` som summerer alle elementene i heltallsarrayen `a`, og som returnerer (det aritmetiske) gjennomsnittet av verdiene i `a`.

Oppgaver til terminaltimen

1. **Løkker og metoder: Fibonacci-tallene**
   Lag et program som skriver ut de 15 første tall i Fibonaccifølgen. Følgen er definert ved at de to første tall er 0 og 1, og hvert neste tall er summen av de to foregående. Utskriften skal bli:

   ```
   0  1  1  2  3  5  8  13  21  34  55  89  144  233  377
   ```

2. **Arrayer, inLine(), inChar(), og enkle metoder:** *(Oblig-relevante!)*
   (Samme oppgaver som i punkt 3., 4., 5., og 6. for teoritimen.)

3. **Start med Oblig 2:**
   Tips: Begynn med å løse foregående oppgave («Arrayer, inLine(), ...»). Der vil du finne problemstillinger som ligner mye på det som skal gjøres i Oblig 2, men litt enklere slik at obligen blir lettere å gjøre etter at du har løst disse oppgavene.

4. **Metoder med inn- og ut-parametre:**
   (Samme oppgaver som i punkt 7. og 8. for teoritimen.) Slike metoder trengs ikke i oblig 2.

5. **Ekstraoppgave:**
   (a) Endre strukturen i Fibonacci-programmet du lagde ovenfor slik at det følger malen fra oblig 2 (vist også ovenfor i punkt 6. for teoritimen), med en kontrollklasse øverst, etterfulgt av en hjelpeklasse for metodene. Bruk tre metoder i hjelpeklassen: en ordreløkke som skriver ut følgende meny, og en metode for hvert av de 2 menyvalgene:

   ```
   1. Skriv ut de 15 første tall i Fibonaccifølgen
   2. Test om et tall hører til følgen
   ```

   Metoden for ordre 2 ber bruker taste inn et tall, og går så i en løkke som genererer Fibonacci-tallene frem til det bruker-inntastede tallet. Deretter gis det melding til bruker om tallet hørte til følgen eller ikke. ***Mer info:*** Fibonacci-tallene forekommer mye i naturen, bl.a. i tregrener, blomster, kongler og kaniner.

   (b) (middels vanskelig) Utvid ordre 2 slik at den også sier hvilket Fibonacci-tall er nærmest det bruker-inntastede tallet.

6. **Ukens nøtt:** *(veldig vanskelig!)*
   Lag en metode som skriver ut alle anagrammer av et ord på 4 bokstaver som ligger i en `char`-array. Anagrammene skal ha de samme 4 bokstavene, i alle mulige rekkefølger og uten å gjenta noen bokstav. For eksempel, hvis ordet er deklarert som følger, er 4 av anagrammene som vist under, og totalt 24.

   ```
   char[] ord = { 'A', 'R', 'N', 'E' };

    Kjøreeksempel:
   ARNE
   AREN
   ANRE
   ANER
   ...20 ord til...
   ```

   Tips: En måte å løse dette på er med nestede løkker som i utgangspunktet kan gå innom alle mulige kombinasjoner, inkludert `AAAA`, `AAAE`, osv. men slik at det bare blir utskrift av de med 4 forskjellige bokstaver. Bruk den tomme strengen `""` og `+` i utskriftssetningen for å konvertere `char`-ene til tekst: `System.out.println("" + ord[3] + ord[2] ...`

Løsningsforslag

Her kan du finne løsningsforslag til disse oppgavene.

Kommentarer om dette oppgavesettet kan du sende til josek *[at]* ifi.uio.no

---

Redaksjon: Redaksjon for informasjon om studietilbudet ved UiO
Dokument opprettet: 08.09.2009, endret: 14.09.2009