INF1000 (Uke 5)
Mer om løkker, arrayer og metoder

Grunnkurs i programmering

Institutt for Informatikk

Universitetet i Oslo

Anja Bråthen Kristoffersen og Are Magnus Bruaset

Praktisk informasjon

- Når disse timene er over har du lært nok til å løse oblig 2
  - Frist 24. februar kl. 16.00.
  - Start tidlig, det er mye å gjøre.

I dag

- Rep: While-løkker
- Mer om løkker: **`do-while`** og **`for`**
- Arrayer
- Litt om metoder

Repetisjon: While-løkker

```
while (logisk uttrykk) {
  //programsetninger
}
```

Variant av while: do-while

- do-while løkke:

```
do {
   //programsetninger
} while (logisk uttrykk);
```

- Noen foretrekker denne fremfor while-løkker.
- I motsetning til for while-løkker vil do-while-løkker alltid utføres minst en gang.

Oppgave

- Lag et oppsett for valg fra meny.
- Skriv ut menyen:

  Velg en av bokstavene A, B eller Q. Velger du:

  A: kjøres metoden A

  B: kjøres metoden B

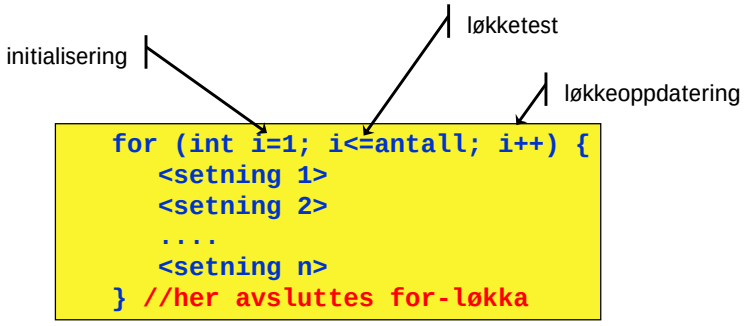
  Q: Avslutter programmet
- Les inn valget, utfør valget og skriv ut menyen pånytt.

Eksempel på do-while løkke

```
do{
        skjerm.outln("Velg en av bokstavene A, B eller Q. Velger du: ");
        skjerm.outln("A: kjøres metoden A");
        skjerm.outln("B: kjøres metoden B");
        skjerm.outln("Q: Avslutter programmet");
        skjerm.out("Valg: ");
        valg = tast.inChar();
        tast.inLine();
        switch(valg){
                case 'A':
                        // her skal jeg kalle på metodeA, lærer det senere i timen;
                        break; // etter break hopper vi ut av switch og tilbake i do-while løkka
                case 'B':
                        // her skal jeg kalle på metodeB, lærer det senere i timen;
                        break;
                case 'Q':
                        skjerm.outln("Programmet avslutter");
                        break;
                default:
                        skjerm.outln("Ugyldig valg");
                        break;
        } // avslutter switch kommandoen.
}while(valg != 'Q');   //her avsluttes do-while løkken.
```


For-setninger

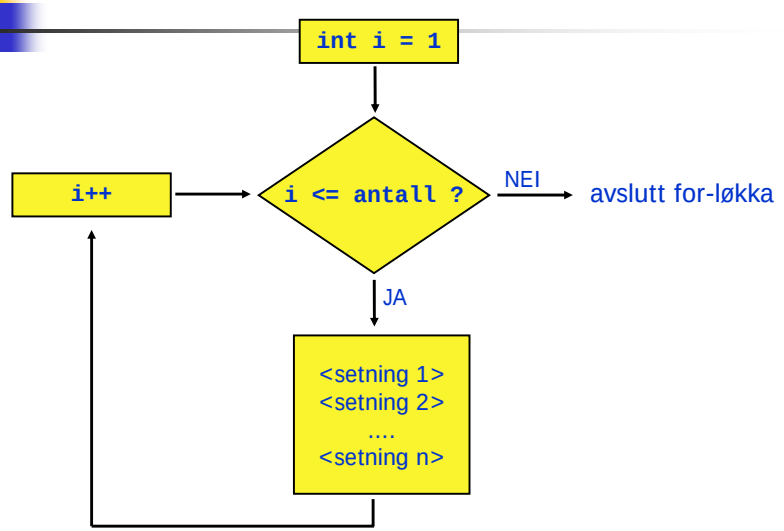
- En annen måte å få utført en instruksjon (evt. blokk av instruksjoner) mange ganger er ved hjelp av en for-løkke:



For-setningens struktur

1. Først deklareres tellevariabelen **`i`**, som settes lik 1
2. Hvis **`i <= antall`** gå til punkt 3; hvis ikke avsluttes for-setningen
3. Utfør setningene inni for-løkke-bolken
4. Utfør **`i++`** og gå til punkt 2

Hvordan for-setningen virker



Operatorene ++ og --

Instruksjon	Alternativ 1 Prefiks-operator	Alternativ 2 Postfiks-operator
`i = i + 1`	`++i`	`i++`
`i = i - 1`	`--i`	`i--`

Prefiks/postfiks - forskjeller

- **`++i, i++, --i`** og **`i--`** endrer ikke bare på verdien til **`i`**, de er dessuten uttrykk som selv har en verdi. Eksempel:

```
// Skriv ut i og øk deretter i med 1 (postfiks)
System.out.println(i++);
// Øk i med 1 og skriv deretter ut i (prefiks)
System.out.println(++i);
```

- Prefiks-operatorene endrer verdien til variabelen før uttrykket er evaluert.
- Postfiks-operatorene endrer verdien etter at uttrykket er evaluert.

Oppgave

Program kode	Verdien til k	Verdien til m	Verdien til n
`int k = 0;`	0	-	-
`int m;`	0	-	-
`int n;`	0	-	-
`k = k + 1;`	1	-	-
`m = ++k;`	2	2	-
`n = 1 + k++;`	3	2	3

Nøtt

- Hva blir utskriften fra dette programmet?

```
class InkrementOperator {
  public static void main (String [] args)
  {
     for (int i = 0; i < 10; i++) {
       int j = i;
       int k = j++ + ++j;
       System.out.println("k = " + k);
     } //slutt for
  } //slutt main
} //slutt InkrementOperator
```

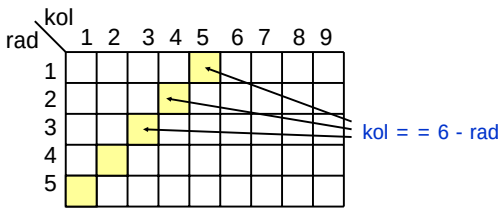

Kompilering og kjøring

```
> javac InkrementOperator.java
➢java InkrementOperator
k=2
k=4
k=6
k=8
k=10
k=12
k=14
k=16
k=18
k=20
```


Nesting av løkker

- Ofte behov for å neste løkke-setninger inne i hverandre
- Eksempel: Skriv følgende på skjermen:

```
1
212
32123
4321234
543212345
```



- Ytre løkke styrer utskrift av linjene
- Indre løkke skriver ut de enkelte tegnene på en linje

Programmet

```
class SkrivPyramide {
   public static void main (String [] args) {
     for (int rad = 1; rad < 6; rad++) {

        for (int kol = 1; kol < 6 - rad; kol++) {
          System.out.print(" ");
        }

        for (int sif = rad; sif >= 1; sif--) {
          System.out.print(sif);
        }

        for (int sif = 2; sif <= rad; sif++) {
          System.out.print(sif);
        }

        System.out.println();
      } //avslutt for (int rad…
   } //avslutt main
} //avslutt SkrivPyramide
```

Kompilering og kjøring

```
> javac SkrivPyramide.java
➢java SkrivPyramide
    1
   212
  32123
 4321234
543212345
```


Variabler

- Hittil har vi sett på variable som kan inneholde en enkelt verdi:
  - en **`int`**-variabel har plass til ett heltall
  - en **`double`**-variabel har plass til ett desimaltall
  - en **`String`**-variabel har plass til en enkelt tekststreng
  - osv.

Arrayer

- Arrayer er "variable" som kan holde på mange verdier:
  - en **`int`**-array har plass til mange heltall
  - en **`double`**-array har plass til mange desimaltall
  - en **`String`**-array har plass til mange tekststrenger
  - osv.

Arrayer

- Verdiene som ligger i en array har hver sin posisjon (= indeks):
  - 0, 1, 2, ..., k-1 hvor k = lengden til arrayen
- En array med lengde 4 kan visualiseres slik:

Deklarere og opprette arrayer

- Syntaks for å deklarere en array:

```
datatype[] arrayNavn;
```

f.eks. int, double, boolean eller String

- Syntaks for å opprette en array. Før vi kan begynne å bruke en deklarert array, må vi fortelle hvor lang den skal være. Det kan gjøres på to måter:

```
int k = 10;
arrayNavn = new datatype[k];  // k er ønsket lengde

arrayNavn = new datatype[]{verdi1, verdi2, …, verdiK};
```


Deklarere og opprette arrayer

- De to trinnene kan slås sammen til en setning:

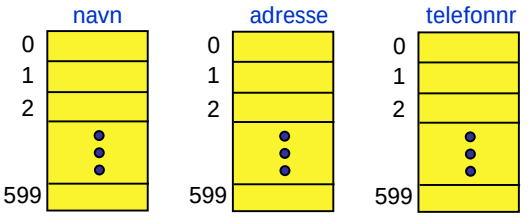
```
int k = 100;
datatype[] arrayNavn = new datatype[k];

datatype[] arrayNavn = {verdi1, …, verdiK};
```


Eksempel

- Ønsker å lagre navn, adresse og telefonnr for studentene på inf1000 (anta max 600 studenter):

```
String[] navn = new String[600];

String[] adresse = new String[600];

int[] telefonnr = new int[600];
```

- Resultatet kan visualiseres slik:



Verdiene i en array

- Anta at vi har deklarert og opprettet følgende array:

```
int[] tlf = new int[600];
```

- For å få tak i de enkelte verdiene i arrayen:

```
tlf[0], tlf[1], tlf[2], ..., tlf[599]
```

- For å få tak i lengden på arrayen:

```
tlf.length   // NB: ingen parenteser til slutt
```


Initialisering av arrayer

- Når en array blir opprettet, blir den samtidig initialisert :
  - `int [] k = new int[100];` // Nå er alle k[i] == 0
  - `double [] x = new double[100];` // Nå er alle x[i] == 0.0
  - `boolean [] b = new boolean[100];` // Nå er alle b[i] == false
  - `char [] c = new char[100];` // Nå er alle c[i] == '\u0000'

Spesielt om String-arrayer

- `String[] s = new String[100];`
  `// Nå er alle s[i] == null`

- Merk: **`String`**-arrayer initialiseres med den spesielle verdien `null`. Dette er *ikke* en tekststreng og må ikke blandes sammen med en tom tekst: `""`.

- For å kunne bruke verdien `s[i]` til noe fornuftig må du først sørge for å gi `s[i]` en tekststreng-verdi, f.eks. `s[i] = "Per"` eller `s[i] = ""`.

Eksempel: Lese 10 desimaltall fra terminal

```
import easyIO.*;

class LesFraTerminal {
  public static void main (String [] args) {
    In tastatur = new In();
    double[] a = new double[10];
    for (int i = 0; i < 10; i++) {
        System.out.print("Oppgi neste verdi: ");
        a[i] = tastatur.inDouble();
    }

    // Nå er verdiene i a lest inn
    // Vi kan evt. gjøre noe med verdiene i a

 } //avslutter main
} //avslutter LesFraTerminal
```

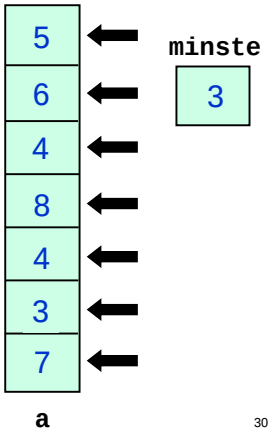
Eksempel: Finne en bestemt verdi

```
/* Gitt int-array a og int-variabel x. Finnes verdien x
også i a? */

   boolean funnet = false;
   int i = 0;
   while (i < a.length && !funnet) {
      if (a[i] = = x) {
         funnet = true;
      }
      i++;
   }
   if (funnet) {
      System.out.println("Verdien ligger i posisjon " + (i-1));
   } else {
      System.out.println("Verdien ble ikke funnet!");
   }
```

Eksempel: Finne minste verdi

- Slik kan man finne den minste verdien i en array:

```
//Anta at x er en double-array

double minste = a[0];
for (int i = 1; i < a.length; i++) {
   if (a[i] < minste) {
      minste = a[i];
   }
}

// Nå er minste lik den minste
// verdien i x
```

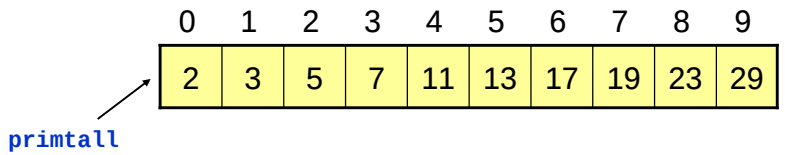


En array-variabel er en adresse

- Når vi deklarerer en array så refererer arraynavnet ikke til selve verdiene i arrayen, men til adressen (i hukommelsen) hvor verdiene ligger lagret.
- Resultatet etter at vi har utført

```
int[] primtall = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
```

kan visualiseres slik:



Å kopiere en array-adresse

- Hvis vi har

```
double[ ] x = new double [100];
```

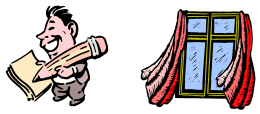
så vil

```
double[ ] y = x;
```

medføre at adressen til arrayen vi opprettet kopieres over til variabelen **y**.

Dermed har vi fortsatt bare ett sett med verdier lagret, men vi har to referanser til arrayen, **x** og **y**.

Oppgave



Hva blir utskriften fra følgende program?

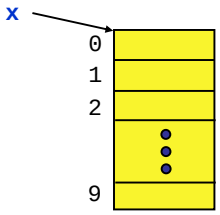
```
class ToArrayer {
  public static void main (String [] args) {
    int[] x = new int[10];
    int[] y = x;

    for (int i = 0; i < 10; i++) {
      x[i] = i;
    }

    for (int i = 0; i < 10; i++) {

      System.out.println(y[i]);
    }
  }
}
```


Kompilering og kjøring

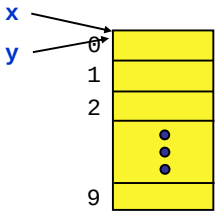
```
> javac ToArrayer.java
> java ToArrayer
0
1
2
3
4
5
6
7
8
9
```


Hva som skjedde...

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




```
int[] y = x;
```



Kopiering av arrayer

Vi kan ikke lage en kopi av en array x ved å skrive

```
int[] y = x;
```

siden dette bare medfører at adressen til arrayen legges inn i y. y peker dermed på samme array sin x peker på.

Kopiering av arrayer

- Skal vi lage en kopi, må vi først opprette en array til (f.eks. y), og så kopiere over verdiene en for en:

```
double[] y = new double[x.length];
for (int i=0; i<x.length; i++) {
  y[i] = x[i];
}
```

- Det finnes også ferdige verktøy i Java for å kopiere en array, f.eks:

```
int [] y = (int []) x.clone();
```


Når arrayen blir for liten

- Kan ha behov for å utvide en array
- Framgangsmåte:
  - Deklarer og opprett en ny array **`temp`** som er av ønsket lengde
  - Flytt over alle verdier fra opprinnelig array til **`temp`**
  - Sett opprinnelig array-peker til å peke på **`temp`**

Når arrayen blir for liten

- Programkode:

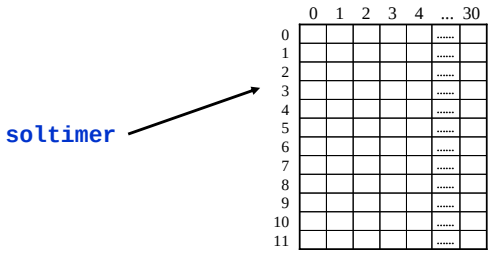
```
/* Anta at tall er en int-array og at vi
ønsker å utvide tall til dobbel lengde */

int[] temp = new int[2 * tall.length];
for (int i = 0; i < tall.length; i++) {
   temp[i] = tall[i];
}
tall = temp;
```


Flerdimensjonale arrayer

- Vi kan også deklarere todimensjonale (og høyeredimensjonale) arrayer.
- Eksempel:

```
int[][] soltimer = new int[12][31];
```

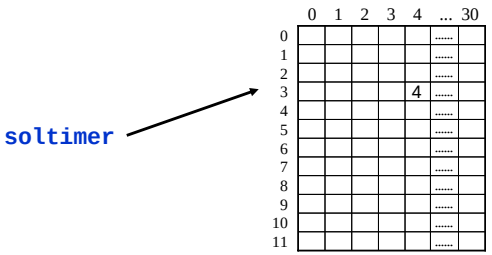
- Resultat:



Flerdimensjonale arrayer

- Eksempler på bruk:

```
soltimer[3][4] = 4;
int antallRader = soltimer.length;
int antallKolonner = soltimer[0].length;
```



Blokker og metoder

- En blokk er en samling instruksjoner omgitt av krøllparenteser:

```
{
  instruksjon 1;
  instruksjon 2;
  ....
  instruksjon n;
}
```

- Alle steder i et Java-program hvor det kan stå en instruksjon, kan vi om ønskelig i stedet sette inn en blokk

Metoder

- Siden en blokk ofte forekommer flere steder i et program, hadde det vært praktisk om vi kunne definert blokken en gang for alle og gitt den et navn.

  Da trenger vi bare å angi blokkens navn hvert sted vi ønsket å få utført instruksjonene i blokken.

- Dette er fullt mulig i Java ved hjelp av det som kalles metoder

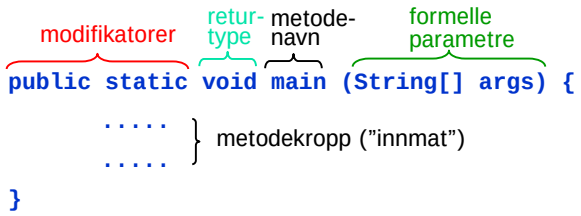
Metoder

- En metode er essensielt en navngitt blokk med instruksjoner som vi kan få utført hvor som helst i et program ved å angi metodens navn

- Beskrivelsen av hva metoden skal hete og hvilke instruksjoner som skal ligge i metoden kalles en metode-deklarasjon.

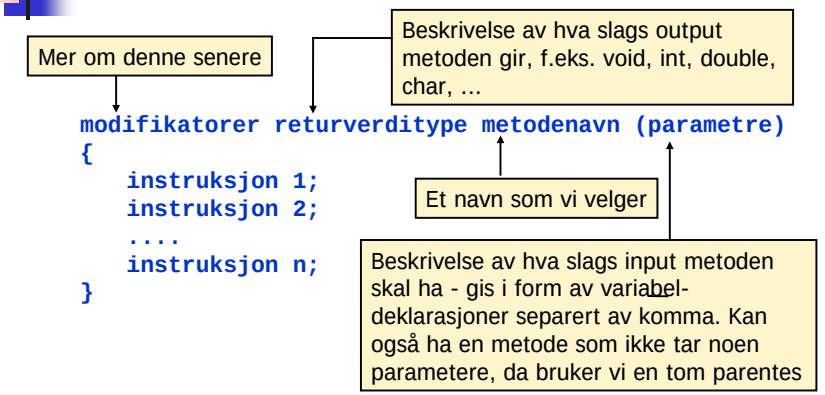
Metoder

- **`main`**-metoden er et eksempel på en metode-deklarasjon:



- En klasse kan inneholde vilkårlig mange metode-deklarasjoner

Å deklarere en metode



Merk at en metode *kan* kreve input og at den *kan* returnere en verdi, men ingen av delene er nødvendig. I enkleste tilfelle er det ingen input og ingen output.

Å benytte en metode

- Når vi benytter en metode sier vi at vi kaller på metoden

- For å kalle på en metode uten parametere, skriver vi ganske enkelt

```
metodenavn();
```


Metoder med parametre

- Ved kall på en metode med parametere må
  - vi oppgi like mange verdier som metoden har parametere
  - i'te verdi må ha samme datatype som i'te parameter i metode-deklarasjonen
- Eksempel:

```
metodenavn(34.2, 53, 6);
```

Metoder med returverdi

- Hvis metoden returnerer en verdi, kan vi velge om verdien skal tas vare på eller ikke når metoden kalles.
- Eksempel hvor vi tar vare på verdien:

```
double sum = metodenavn(25.3, 52, 7);
```


Eksempel: Metode uten input/output

```
static void skrivStjerner() {
   String s = "****************************";
   System.out.println(s);
   System.out.println(s);
   System.out.println(s);
   System.out.println(s);
}
```

- Forklaring:
  - static er en modifikator som forteller at dette er en klassemetode og ikke en objektmetode, dvs metoden skal ikke benyttes inni et objekt.
  - void er en returverditype som forteller at metoden ikke gir noe output.
  - skrivStjerner er det navnet vi har valgt å gi metoden

Eksempel på bruk


```
class Stjerner {

  public static void main (String[] args) {
    skrivStjerner();
    System.out.println("Hei");
    skrivStjerner();
  } //slutt main

  static void skrivStjerner () {
    String s = "****************************";
    System.out.println(s);
    System.out.println(s);
    System.out.println(s);
    System.out.println(s);
  } //slutt skrivStjerner

} //slutt Stjerner
```


Kompilering og kjøring

```
> javac Stjerner.java
> java Stjerner
****************************
****************************
****************************
****************************
Hei
****************************
****************************
****************************
****************************
```