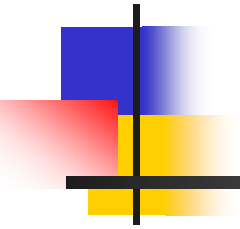


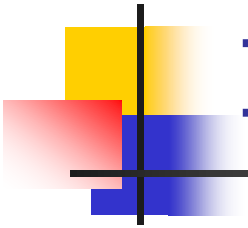
INF 1000 – høsten 2011
Uke 4: 13. september

Grunnkurs i Objektorientert Programmering

Institutt for Informatikk

Universitetet i Oslo

Siri Moe Jensen og Arne Maus



Innhold – uke 4

- Repetisjon m/ utvidelser: Arrayer
  - representasjon av array: arrayreferanse (peker) og arrayobjekt
  - initialisering av arrayer
- Flerdimensjonale arrayer
- Metoder
  - En ny programstruktur
  - deklarasjon og kall
  - returverdier og parametere

Mål for uke 4:

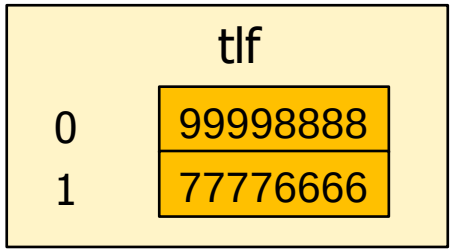
* Java: Flerdimensjonale arrayer, metoder (Kap. 5.7, 7.1-7.7)

* Programmering: Designe og skrive programmer med ny struktur inkl. metoder

Representasjon av array i Java

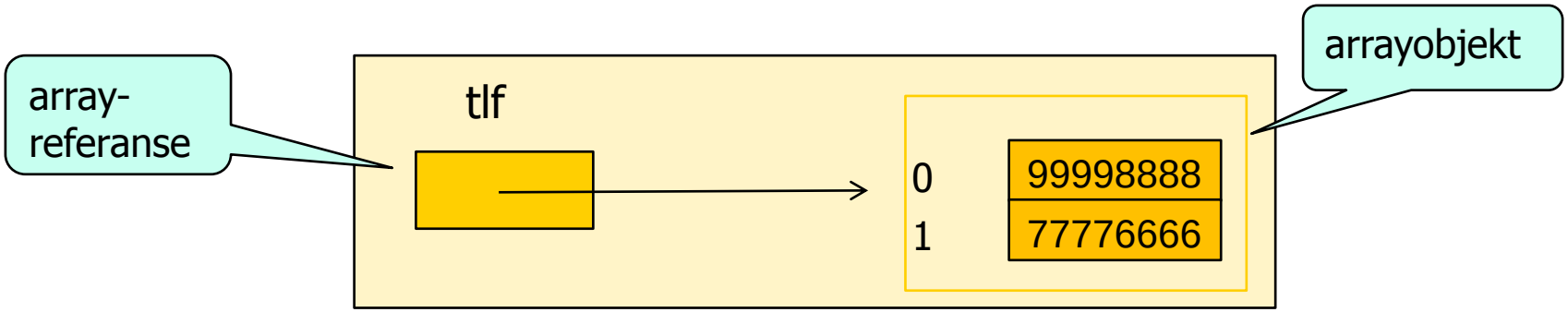
- En array er en rekke med variable av samme type, som aksesseres med arraynavn og indeks

```
int [] tlf = new int [2];
tlf [0] = 99998888;
tlf [1] = 77776666;
```



forenklet tegning

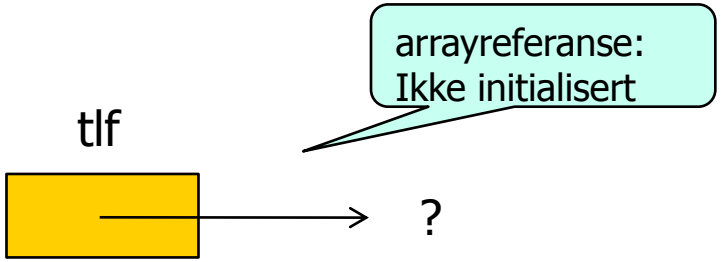
- Mer presist: I Java representeres dette som en *arrayreferanse* til et *arrayobjekt*



Deklarasjon av peker og opprettelse av arrayobjektet

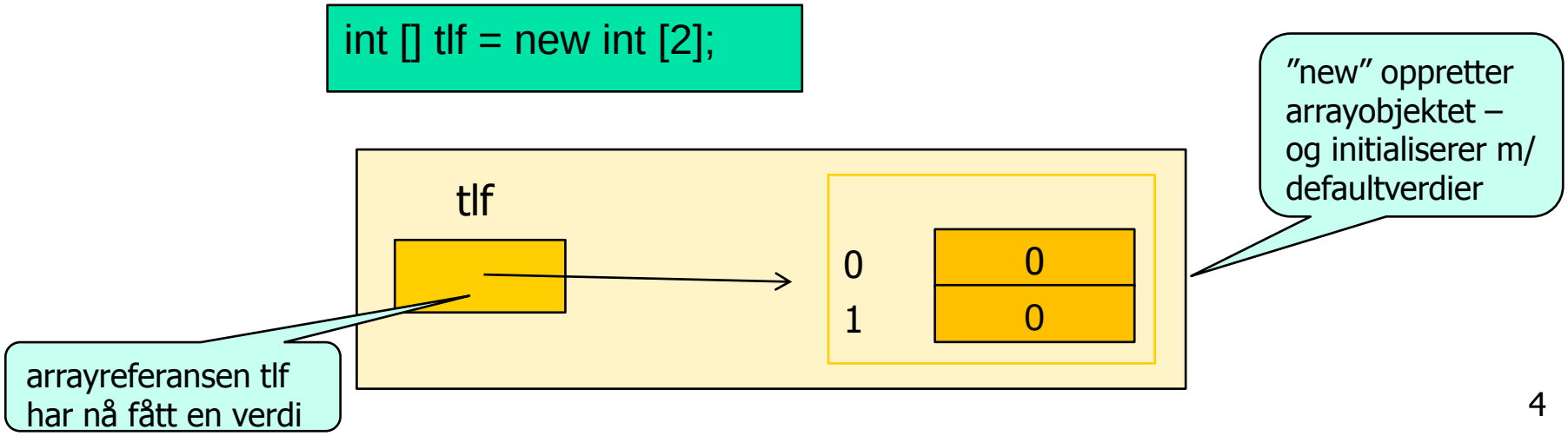
- Deklarasjon av arraypeker

```
int [] tlf;
```



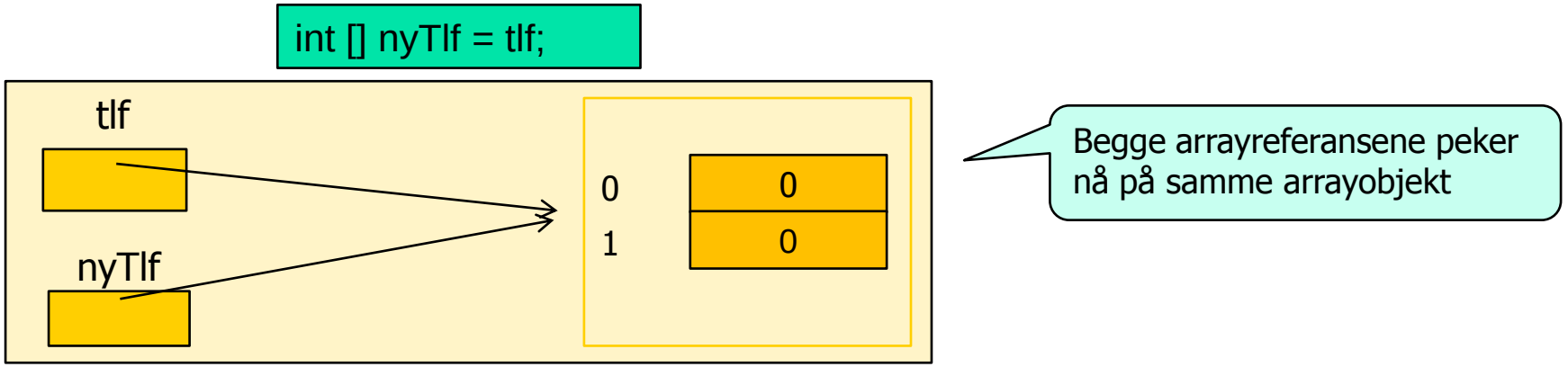
- Arrayobjekt med angitt størrelse opprettes – og pekes på av tlf

```
int [] tlf = new int [2];
```

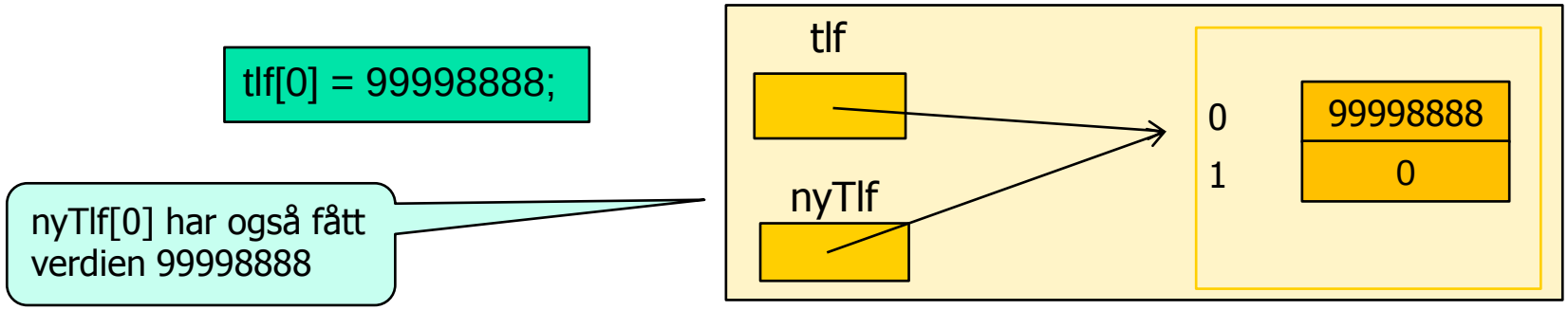


Kopiering av arrayreferanse

- Vi setter en ny arrayreferanse lik den vi har

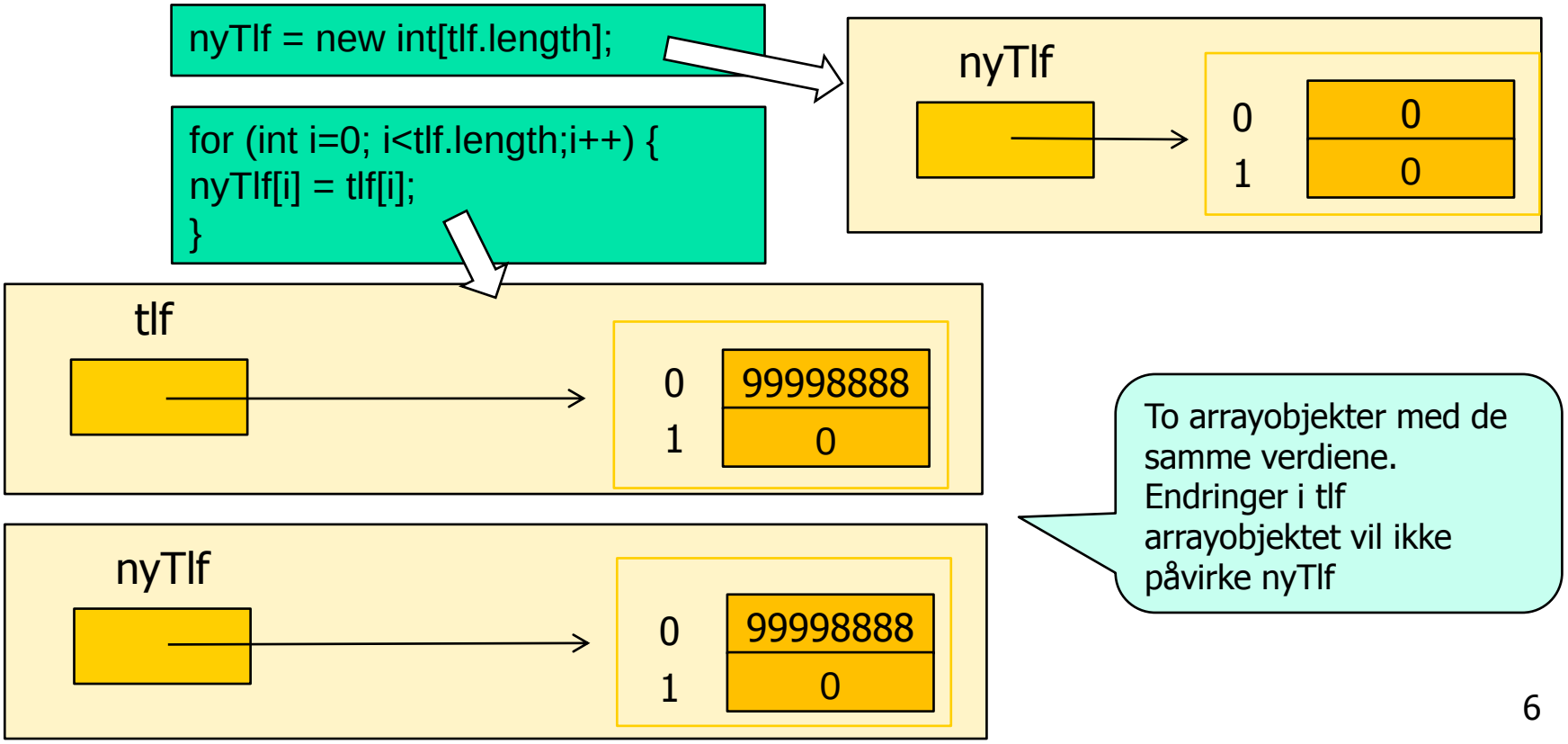


- Hvis vi nå endrer verdiene i arrayobjektet – vil endringene synes uansett hvilken arrayreferanse vi leser av



Kopiering av array

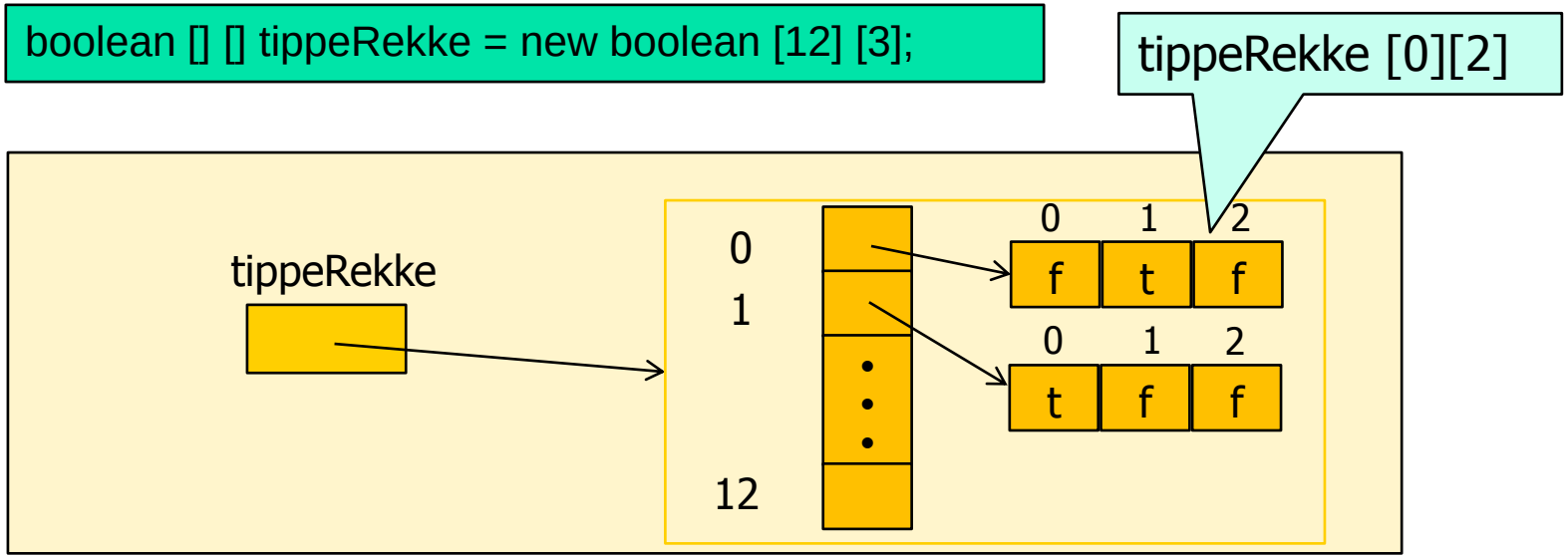
- Skal vi ha en egen kopi av alle plassene i arrayen må vi sette av ny plass med new, og kopiere en og en verdi, f eks. i en for-løkke



Flerdimensjonal array

- Dersom vi ønsker å representere en todimensjonal tabell av verdier *av samme type* kan vi deklarere en todimensjonal array

```
boolean [] [] tippeRekke = new boolean [12] [3];
```



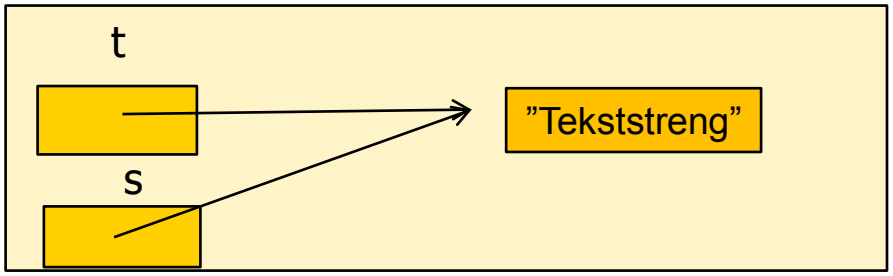
- Dette kan generaliseres for N dimensjoner – her 3:

```
boolean [] [] [] tippeKupong = new boolean [10][12] [3];
```

Kopiering og testing av String-verdier

- String-variable er referanser (pekere) til tekstobjekter

```
String s = ”Tekststreng”;
String t = s;
```

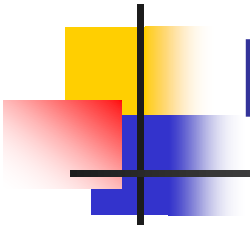



```
(s==t)
(s.equals(t))
(t.equals(s))
```

logiske utsagn – alle **true**

- Tekstobjekter kan imidlertid ikke endres – alle endringer fører til opprettelse av et nytt tekstobjekt, slik at andre tekstvariable som peker på samme objekt ikke endres (forskjellig fra arrayobjekter!)

- Uttrykket (s==t) er bare true dersom de peker på den *samme* tekststrengen. Skal vi teste på om to tekststrenger er *like* (men ikke nødvendigvis den samme) må vi bruke (s.equals(t))



Metoder – hvorfor?

- Ofte har vi bruk for å utføre (omtrent) samme instruksjoner flere steder i et program, eller i mange programmer. Gjenbruk
  - reduserer størrelsen på programmet
  - bedrer oversikten
  - forenkler vedlikeholdet
  - gir færre feil

- Da er det nyttig å kunne samle en eller flere instruksjoner med et selvvalgt navn, som så kan benyttes en eller flere ganger i programmet vårt

Metoder - eksempler

- Vi har brukt en del metoder allerede, for eksempel

```
System.out.println(" * ");
double d = tastatur.inDouble();
String s = tastatur.inWord();
int i = (int) Math.round(d);
```

- Også main() er en metode:

```
public static void main(String[] args) {
.....
}
```

- Vi skal nå se nærmere på hva metoder egentlig er, hvordan de brukes og hvordan vi kan lage våre egne metoder.

Metoder

- En metode er en navngitt blokk med instruksjoner (programsetninger) som vi får utført ved å angi metodens navn
- Den *deklareres* inne i en klasse, og kan senere *kalles på* hver gang vi ønsker den utført
- Eksempel på deklarasjon:

metodens navn

metodens type (hvis returverdi)

```
void skrivPyramide() {
    System.out.println(" * ");
    System.out.println(" *** ");
    System.out.println("*****");
}
```

- Bruk av (kall på) metoden:

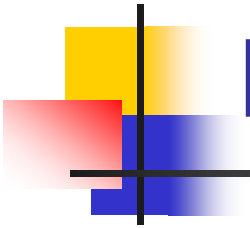
```
skrivPyramide();      // Her utføres setningen i metoden
```

Snartur innom konsepter

Objektorientert programmering

- Programmeringsspråk er designet etter ulike ”paradigmer” eller hovedprinsipper.
- Java er et *objektorientert* språk (andre typer er imperative eller funksjonelle)
- Det betyr at språket spesielt støtter programmereren i å tenke på det som skal beskrives og bearbeides i form av *objekter* – som har både egenskaper (*variable*) og handlinger (programsetninger samlet i *metoder*) knyttet til seg.
- Hvilke variable og metoder som hører til et objekt bestemmes av hvilken *klasse* objektet tilhører – en klasse beskriver et mønster for en type objekter.
- Viktige egenskaper ved objektorienterte programmer er muligheten til å *skjule detaljer* som ikke er vesentlig for omgivelsene (vi velger selv hva som skal synes og brukes fra utsiden av en klasse) og å *gruppere* variable og metoder som hører sammen.

Dette kommer vi tilbake til...



Programstruktur med metoder

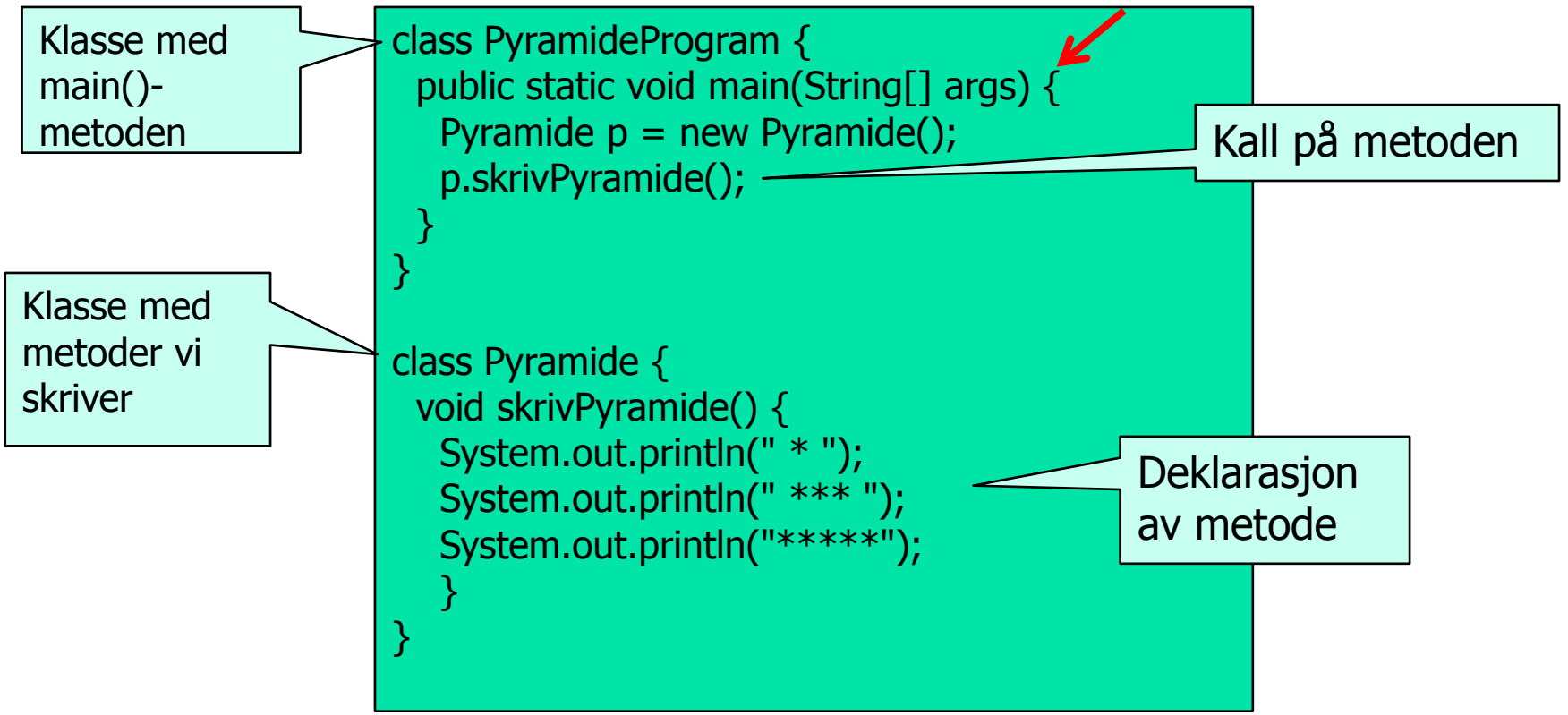
- Alle Java-programmer består av en eller flere klasser, med en eller flere metoder. Metoder deklareres inne i klasser.

- Hittil har programmene våre bestått av én klasse med metoden main() i. Klassen har samme navn som filen, og main() – som er der utføring av programmet starter og slutter - er den eneste metoden vi har skrevet, med all funksjonalitet i seg.

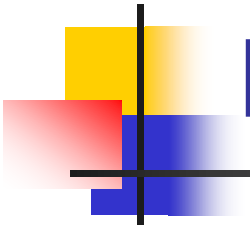
- Dere vil senere lære å skrive programmer med mange klasser og metoder – først skal vi imidlertid konsentrere oss om å lære deklarasjoner og bruk av metoder i en enkel, standard programstruktur
  - En klasse med main()-metoden, der program-utførelsen starter og avsluttes
  - En klasse med datastruktur, og metoder som bearbeider denne

Enkel programstruktur med metoder

- Foreløpig vil vi benytte følgende mal for programmer med metoder (her et program fra filen PyramideProgram.java)



Metode med returverdi



- Ofte ønsker vi at resultatet fra en metode skal kunne brukes videre i resten av programmet, som for eksempel:

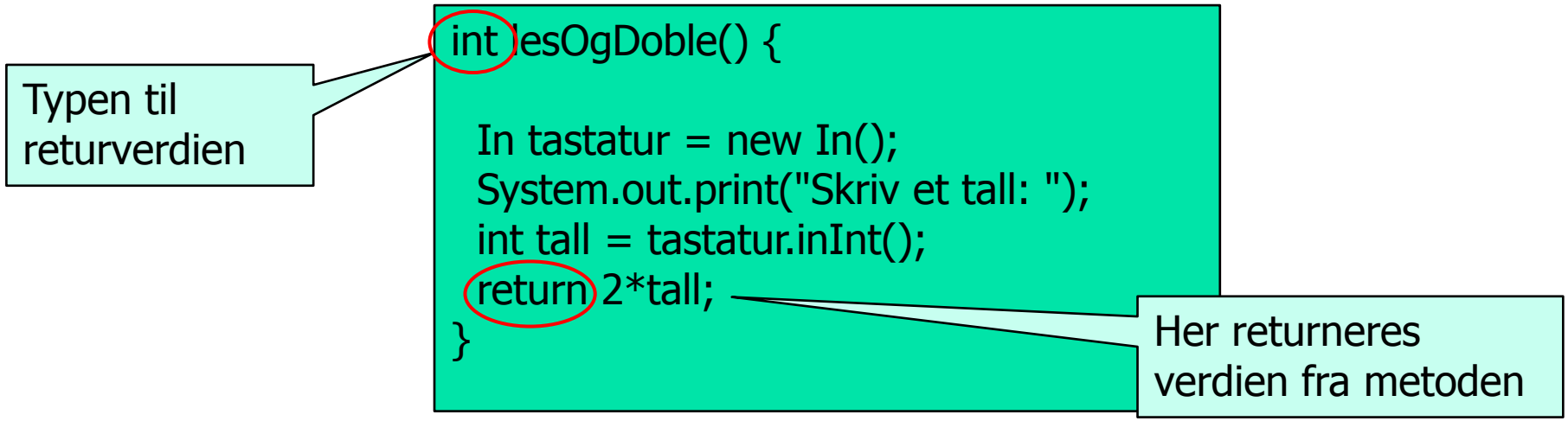
```
In tastatur = new In();

int k = tastatur.inInt();
```

- Her **inInt()** en metode i klassen **In**, som vi får tilgang til via pekervariabelen **tastatur.**
- Metoden **inInt() returnerer** et heltall (en **int**).
- Dette heltallet tar vi vare på i variabelen **k.**

Deklarasjon av metode med returverdi

- Her er en metode som leser et heltall og returnerer det dobbelte:



- Metoder som ikke returnerer noen verdi deklareres med typen **void**.

Metode med returverdi: Eksempel

- Metoden **inInt()** leser inn et vilkårlig heltall. Noen ganger ønsker vi å sikre oss at vi får et **positivt** heltall. Vi kan da lage en egen metode for dette:

```
int lesPositivtHeltall() {

  In tastatur = new In();
  int tall;
  do {
    System.out.print ("Gi et positivt tall: ");
    tall = tastatur.inInt();
  } while (tall <= 0);

  return tall;
}
```

Deklarasjon av metoden

Eks: Program med bruk av metode med returverdi

```
import easyIO.*;
class TestPosTall {
    public static void main(String[] args) {
            PosTall pt = new PosTall();

            int i = pt.lesPositivtHeltall();
            System.out.println(i + " er et positivt heltall");
    }
}
class PosTall {
    int lesPositivtHeltall() {
            In tastatur = new In();
            int tall;
            do {
                System.out.print("Gi et positivt tall: ");
                tall = tastatur.inInt();
            } while (tall <= 0);
            return tall;
    }
}
```

Metoder med parametere

- Ofte ønsker vi at samme metode skal kunne brukes for litt ulike input-verdier, f. eks:

```
System.out.println(" * ");

System.out.println("Hei verden");
```

- Her er println() en metode som tar en tekst som input (**parameter)**

- Metoden gjør det samme uansett hvilken String-verdi vi kaller med: Skriver den ut på skjermen

Eks: Metode med parameter

- Parameteren avgjør antall linjer i ”pyramiden”:

```
class PyramideProgram {
    public static void main(String[] args) {
            Pyramide p = new Pyramide();
            p.skrivPyramide(4);
    }
}
class Pyramide {
    void skrivPyramide(int antall) {

            for (int i = 1; i<=antall; i++) {
                for (int j = 1; j<=i; j++) {
                        System.out.print("*");
                }
                System.out.println ();
            }
    }
}
```

parameter angis med type og navn som skal benyttes inni metoden

Eks: Gangetabell-metode

- Metode som skriver ut n-gangen fra 1 til 10

```
void gangeTabell(int n) {
    for (int i = 1; i <= 10; i++) {
        System.out.println(i * n);
    }
}
```

n

2

- Eksempler på kall på metoden

```
gangeTabell(2);
gangeTabell(15);
```

n

15

Eks: Bruk av gangetabell-metode

```
import easyIO.*;
class GangeProgram {
    public static void main(String[] args) {
            In tastatur = new In();
            Utregning utr = new Utregning();
            System.out.print("Hvilken gangetabell vil du skrive? ");
            int tall = tastatur.inInt();
            utr.gangeTabell(tall);
    }
}
class Utregning {
    void gangeTabell(int n) {
            for (int i = 1; i <= 10; i++) {
                System.out.println(i * n);
            }
    }
}
```

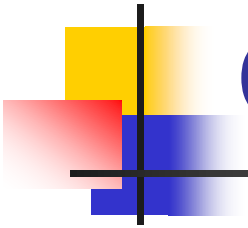
Eksempel med flere parametre

- Vi kan utvide gangeTabell-metoden med en parameter som angir hvor mye av tabellen som skal skrives:

```
void gangeTabell(int n, int slutt) {
    for (int i = 1; i <= slutt; i++) {
        System.out.println(i * n);
    }
}
```

- Eksempel på kall på metoden:

```
gangeTabell(2, 20);
gangeTabell(15, 10);
```



Oppgave

- Skriv et program som deklarerer og bruker denne metoden.
- Initialiser arrayen selv – skriv ut summen.

```
double finnSum (double[] x) {
   double sum = 0.0;
   for (int i=0; i<x.length; i++) {
      sum += x[i];
   }
   return sum;
}
```

Oppgave - løsningsforslag

```
class TestMetode {
    public static void main (String [] args) {
            double [] a = {1.3, 2.0, 7.5, 10};

            MetodeKlasse mk = new MetodeKlasse ();
            double total = mk.finnSum(a);
            System.out.println ("Returverdi = " + total);
    }
}

class MetodeKlasse {
    double finnSum(double[] x) {
            double sum = 0.0;
            for (int i=0; i<x.length; i++) {
                sum += x[i];
            }
            return sum;
    }
}
```

Oppsummering metoder: Deklarasjon

- Java-programmene så langt i kurset består av to klasser, startklassen med main og en annen klasse hvor det kan det befinne seg en eller flere metoder.
- De metodene vi ser på så langt i kurset har følgende form:

type verdi metoden returnerer, f.eks. int, double, char, ... eller void hvis ingen verdi

et navn vi velger

Beskrivelse av hva slags input metoden skal ha - i form av variabel-deklarasjoner skilt av komma

```
returverditype metodenavn (parametre) {
        setninger 1;
        setninger 2;
        ....
        setninger n;
        return verdi;    // Hvis ikke returverditype void
}
```

Oppsummering metoder: Kall

- Når vi benytter en metode sier vi at vi *kaller på* metoden
- Kall på metode uten parametere – eks:

```
minMetode();
```

- Kall på metode med parametere – eks:

```
minMetode2 (2, ”Hei!”);
```

- Kall på metode med parameter som returnerer en verdi – eks:

```
int i = minMetode3 (10);
```

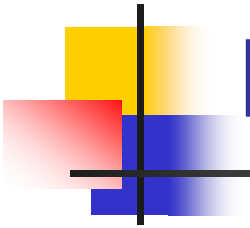
Parametre og argumenter

```
class MittProgram {
    public static void main(String[] args) {
            Kalkulator k = new Kalkulator();
            double pris = 100.0;
            double nyPris = k.trekkFraRabatt(pris);
            System.out.println("Utsalgspris: " + nyPris);
    }
}
class Kalkulator {
    double trekkFraRabatt(double x) {
            return x * 0.8;
    }
}
```

Argument

Parameter

- Merk: argumenter til metodekallet kalles også for *aktuelle parametre* mens parametre i deklarasjonen da kalles *formelle parametre*.



Parametre og argumenter

- Parametre
  - Deklareres mellom parentesene i toppen (= den første linjen) av metode-deklarasjonen. De er ”vanlige variable” som bare eksisterer inne i metoden og så lenge denne eksekverer.
- Argumenter
  - Verdier som oppgis mellom parentesene når vi kaller på en metode.
  - Antall argumenter må samsvare med antall parametre i metoden
  - Argumentenes datatyper må samsvare med datatypen til tilsvarende parameter.