

# Utførelse av Java-kode

- De fleste har en intuitiv forståelse av mer eller mindre enkle Java-programmer (variabeldeklarasjoner, uttrykk, metodekall osv.)
- Imidlertid trengs en mer detaljert modell for at nyanser og kompliserte tilfeller skal kunne forstås.
- Eksempel:
  - ```
for (int i = 0; i < 10; i++) {
    if (...) {
        break;
    }
}
```

 hvilken verdi har i-variablen etter for-setningen?
- Eksempel:
  - ```
public int foo(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return foo(n - 1) + foo(n - 2);
    }
}
```

 foo-metoden kaller seg selv, to ganger til og med, blir det ikke da evig nøsting?

1

1

## Manuell eller mental utførelse av kode

- Veldig viktig å kunne forstå hvordan kode blir utført
  - andres kode: hva gjør koden som en skal (gjen)bruke
  - testkode: hvordan tester egentlig denne koden min kode
  - egen kode (!): hvorfor gjør koden min noe annet enn det jeg tror
- Studenter spør ofte: “Hvorfor virker ikke koden min?” Mange mulige svar:
  - “Java gjør det den blir bedt om!”
  - “Det burde du vite som har skrevet den!”
  - “Har du ikke lest koden din selv...”
  - (Jeg har det innimellom sånn også, selv på forelesning...)
- Du må alltid lese koden din og tenke gjennom om den gjør det du ønsker at den skal gjøre!
  - Husk også hvor kraftig debuggeren er

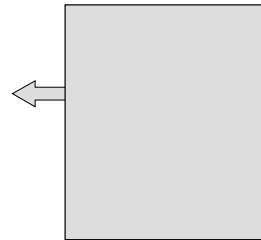
2

2

## ”Ark”-modellen for kjøring

- Tenk på programmet som en oppskrift med et tilhørende ark for å notere verdier.
- Til arket hører en pil som peker på neste linje/setning i programmet.
- Hver gang en kommer til en variabeldeklarasjon, utvides arket med en navngitt verdi.
- Ved tilordning, erstattes den gamle verdien med den nye.
- Eksempel:

```
int a = 1;  
int b = a + 1;  
a = b + 1;
```



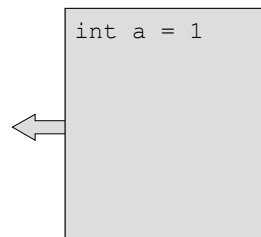
3

3

## ”Ark”-modellen for kjøring

- Tenk på programmet som en oppskrift med et tilhørende ark for å notere verdier.
- Til arket hører en pil som peker på neste linje/setning i programmet.
- Hver gang en kommer til en variabeldeklarasjon, utvides arket med en navngitt verdi til.
- Ved tilordning, erstattes den gamle verdien med den nye.
- Eksempel:

```
int a = 1;  
int b = a + 1;  
a = b + 1;
```



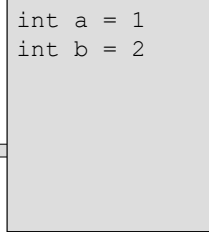
4

4

## ”Ark”-modellen for kjøring

- Tenk på programmet som en oppskrift med et tilhørende ark for å notere verdier.
- Til arket hører en pil som peker på neste linje/setning i programmet.
- Hver gang en kommer til en variabeldeklarasjon, utvides arket med en navngitt verdi til.
- Ved tilordning, erstattes den gamle verdien med den nye.
- Eksempel:

```
int a = 1;  
int b = a + 1;  
a = b + 1;
```



```
int a = 1  
int b = 2
```

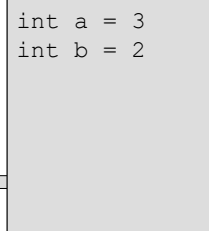
5

5

## ”Ark”-modellen for kjøring

- Tenk på programmet som en oppskrift med et tilhørende ark for å notere verdier.
- Til arket hører en pil som peker på neste linje/setning i programmet.
- Hver gang en kommer til en variabeldeklarasjon, utvides arket med en navngitt verdi til.
- Ved tilordning, erstattes den gamle verdien med den nye.
- Eksempel:

```
int a = 1;  
int b = a + 1;  
a = b + 1;
```



```
int a = 3  
int b = 2
```

6

6

## ”Ark”-modellen forts.

- Hver type Java-”snutt” kan forklares ved å vise hvordan ”arket” brukes eller endres av at snutten kjøres.
- Deklarasjon: `<type> <navn>;`
  - F.eks. `int a;`
  - ny variabel legges til arket og settes til standardverdien for typen
- Deklarasjon: `<type> <navn> = <val>;`
  - F.eks. `int a = 1;`
  - ny variabel legges til arket
- Tilordning: `<navn> = <val>;`
  - F.eks. `a = 2;`
  - verdien til variabelen erstattes
- Tilleggsregel:  
Sammensatte deklarasjoner gjøres i sekvens:
  - `int a, b; utføres som`  
`int a; int b;`

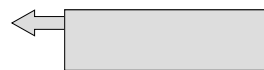
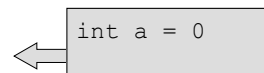


7

7

## ”Ark”-modellen forts.

- Hver type Java-”snutt” kan forklares ved å vise hvordan ”arket” brukes eller endres av at snutten kjøres.
- Deklarasjon: `<type> <navn>;`
  - F.eks. `int a;`
  - ny variabel legges til arket og settes til standardverdien for typen
- Deklarasjon: `<type> <navn> = <val>;`
  - F.eks. `int a = 1;`
  - ny variabel legges til arket
- Tilordning: `<navn> = <val>;`
  - F.eks. `a = 2;`
  - verdien til variabelen erstattes
- Tilleggsregel:  
Sammensatte deklarasjoner gjøres i sekvens:
  - `int a, b; utføres som`  
`int a; int b;`

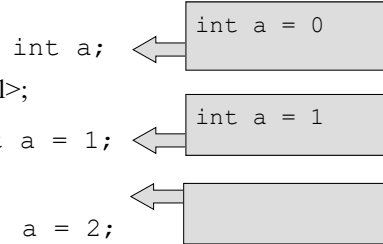


8

8

## ”Ark”-modellen forts.

- Hver type Java-”snutt” kan forklares ved å vise hvordan ”arket” brukes eller endres av at snutten kjøres.
- Deklarasjon: `<type> <navn>;`
  - F.eks. `int a;`
  - ny variabel legges til arket og settes til standardverdien for typen
- Deklarasjon: `<type> <navn> = <val>;`
  - F.eks. `int a = 1;`
  - ny variabel legges til arket
- Tilordning: `<navn> = <val>;`
  - F.eks. `a = 2;`
  - verdien til variabelen erstattes
- Tilleggsregel:  
Sammensatte deklarasjoner gjøres i sekvens:
  - `int a, b; utføres som`  
`int a; int b;`

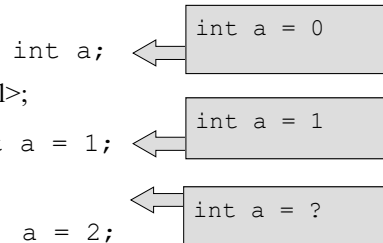


9

9

## ”Ark”-modellen forts.

- Hver type Java-”snutt” kan forklares ved å vise hvordan ”arket” brukes eller endres av at snutten kjøres.
- Deklarasjon: `<type> <navn>;`
  - F.eks. `int a;`
  - ny variabel legges til arket og settes til standardverdien for typen
- Deklarasjon: `<type> <navn> = <val>;`
  - F.eks. `int a = 1;`
  - ny variabel legges til arket
- Tilordning: `<navn> = <val>;`
  - F.eks. `a = 1;`
  - verdien til variabelen erstattes
- Tilleggsregel:  
Sammensatte deklarasjoner gjøres i sekvens:
  - `int a, b; utføres som`  
`int a; int b;`

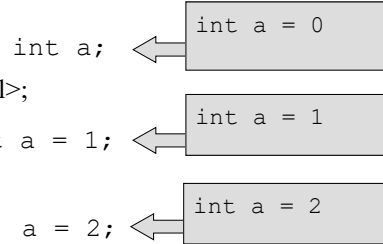


10

10

## ”Ark”-modellen forts.

- Hver type Java-”snutt” kan forklares ved å vise hvordan ”arket” brukes eller endres av at snutten kjøres.
- Deklarasjon: `<type> <navn>;`
  - F.eks. `int a;`
  - ny variabel legges til arket og settes til standardverdien for typen
- Deklarasjon: `<type> <navn> = <val>;`
  - F.eks. `int a = 1;`
  - ny variabel legges til arket
- Tilordning: `<navn> = <val>;`
  - F.eks. `a = 2;`
  - verdien til variabelen erstattes
- Tilleggsregel:  
Sammensatte deklarasjoner gjøres i sekvens:
  - `int a, b;` utføres som  
`int a; int b;`



11

11

## ”Ark” kan ligge i lag og komme og gå

- Med `{ ... }` vil et nytt, temporært ark kan legges oppå et eksisterende.
- Eksempel:
 

```
int a = 1;
{
    int b = a + 1;
    a = b + 1;
}
```
- Slike `{...}`-blokker er et signal om at variablene har kortvarig relevans, f.eks. inneholder midlertidige mellomverdier.
- Nye (temporære) variabler kan introduseres i alle slike `{ ... }`-blokker, også i if-else-grenser og i while og for-løkker.
- Når en blokk er utført, fjernes arket og tilhørende variabler.

12

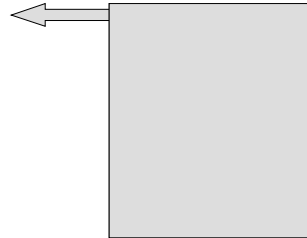
12

## ”Ark” kan ligge i lag og komme og gå

- Med { . . . } vil et nytt, temporært ark kan legges oppå et eksisterende.

Eksempel:

```
int a = 1;
{
    int b = a + 1;
    a = b + 1;
}
```



- Nye (temporære) variabler kan introduseres i alle slike { . . . }-blokker, også i if-else-grenser og i while og for-løkker.
- Når variabler utføres som uttrykk eller tilordnes en ny verdi, letes variablen opp på arkene som er aktive, fra øverst til underst.
- Når en blokk er utført, fjernes arket og tilhørende variabler.

13

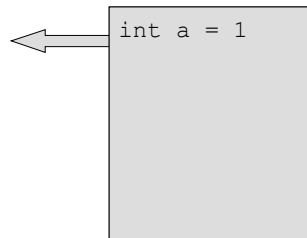
13

## ”Ark” kan ligge i lag og komme og gå

- Med { . . . } vil et nytt, temporært ark kan legges oppå et eksisterende.

Eksempel:

```
int a = 1;
{
    int b = a + 1;
    a = b + 1;
}
```



- Nye (temporære) variabler kan introduseres i alle slike { . . . }-blokker, også i if-else-grenser og i while og for-løkker.
- Når variabler utføres som uttrykk eller tilordnes en ny verdi, letes variablen opp på arkene som er aktive, fra øverst til underst.
- Når en blokk er utført, fjernes arket og tilhørende variabler.

14

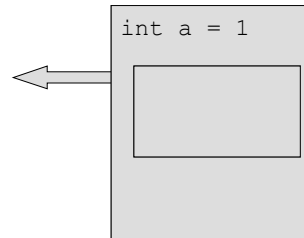
14

## ”Ark” kan ligge i lag og komme og gå

- Med `{ ... }` vil et nytt, temporært ark kan legges oppå et eksisterende.

Eksempel:

```
int a = 1;
{
    int b = a + 1;
    a = b + 1;
}
```



- Nye (temporære) variabler kan introduseres i alle slike `{ ... }`-blokker, også i if-else-grenser og i while og for-løkker.
- Når variabler utføres som uttrykk eller tilordnes en ny verdi, letes variablen opp på arkene som er aktive, fra øverst til underst.
- Når en blokk er utført, fjernes arket og tilhørende variabler.

15

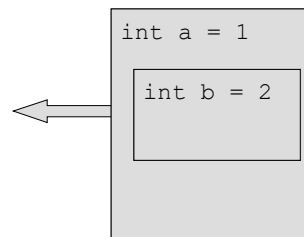
15

## ”Ark” kan ligge i lag og komme og gå

- Med `{ ... }` vil et nytt, temporært ark kan legges oppå et eksisterende.

Eksempel:

```
int a = 1;
{
    int b = a + 1;
    a = b + 1;
}
```



- Nye (temporære) variabler kan introduseres i alle slike `{ ... }`-blokker, også i if-else-grenser og i while og for-løkker.
- Når variabler utføres som uttrykk eller tilordnes en ny verdi, letes variablen opp på arkene som er aktive, fra øverst til underst.
- Når en blokk er utført, fjernes arket og tilhørende variabler.

16

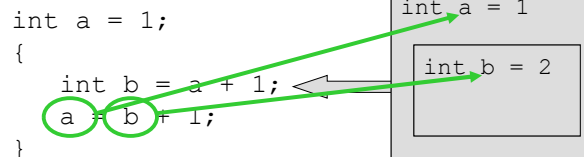
16



## ”Ark” kan ligge i lag og komme og gå

- Med `{ ... }` vil et nytt, temporært ark legges oppå et eksisterende.

- Eksempel:



- Nye (temporære) variabler kan introduseres i alle slike `{ ... }`-blokker, også i if-else-grenser og i while og for-løkker.
- Når variabler utføres som uttrykk eller tilordnes en ny verdi, letes variablen opp på arkene som er aktive, fra øverst til underst.
- Når en blokk er utført, fjernes arket og tilhørende variabler.

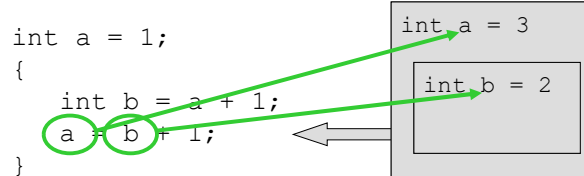
17

17

## ”Ark” kan ligge i lag og komme og gå

- Med `{ ... }` vil et nytt, temporært ark legges oppå et eksisterende.

- Eksempel:



- Nye (temporære) variabler kan introduseres i alle slike `{ ... }`-blokker, også i if-else-grenser og i while og for-løkker.
- Når variabler utføres som uttrykk eller tilordnes en ny verdi, letes variablen opp på arkene som er aktive, fra øverst til underst.
- Når en blokk er utført, fjernes arket og tilhørende variabler.

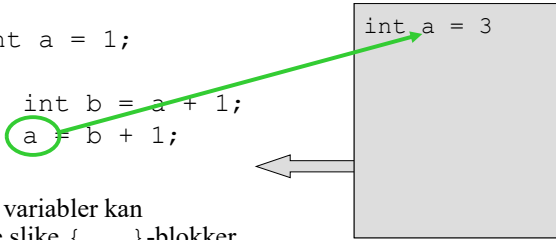
18

18

## ”Ark” kan ligge i lag og komme og gå

- Med `{ . . . }` vil et nytt, temporært ark kan legges oppå et eksisterende.
- Eksempel:
 

```
int a = 1;
{
    int b = a + 1;
    a = b + 1;
}
```


- Nye (temporære) variabler kan introduseres i alle slike `{ . . . }`-blokker, også i if-else-grenser og i while og for-løkker.
- Når variabler utføres som uttrykk eller tilordnes en ny verdi, letes variablen opp på arkene som er aktive, fra øverst til underst.
- Når en blokk er utført, fjernes arket og tilhørende variabler.


19

19

## while

- while**-setningen endrer ikke arket, men spesifiserer regler for hvordan pilen flytter seg
  - testen og kroppen utføres vekselvis, til testen gir false som verdi
- Eksempel:

```
int a = 1;
while (a < 4) {
    a += 2;
}
```



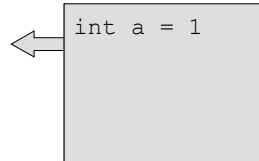
20

20

# while

- **while**-setningen endrer ikke arket, men spesifiserer regler for hvordan pilen flytter seg
  - testen og kroppen utføres vekselvis, til testen gir false som verdi
- Eksempel:

```
int a = 1;
while (a < 4) {
    a += 2;
}
```



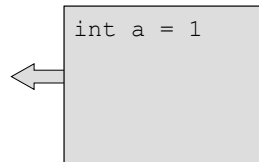
21

21

# while

- **while**-setningen endrer ikke arket, men spesifiserer regler for hvordan pilen flytter seg
  - testen og kroppen utføres vekselvis, til testen gir false som verdi
- Eksempel:

```
int a = 1;
while (a < 4) {
    a += 2;
}
```



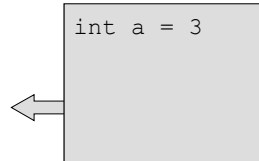
22

22

# while

- **while**-setningen endrer ikke arket, men spesifiserer regler for hvordan pilen flytter seg
  - testen og kroppen utføres vekselvis, til testen gir false som verdi
- Eksempel:

```
int a = 1;  
while (a < 4) {  
    a += 2;  
}
```



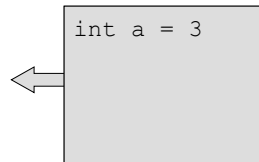
23

23

# while

- **while**-setningen endrer ikke arket, men spesifiserer regler for hvordan pilen flytter seg
  - testen og kroppen utføres vekselvis, til testen gir false som verdi
- Eksempel:

```
int a = 1;  
while (a < 4) {  
    a += 2;  
}
```



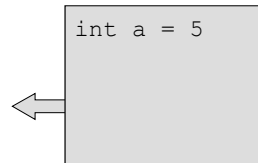
24

24

# while

- **while**-setningen endrer ikke arket, men spesifiserer regler for hvordan pilen flytter seg
  - testen og kroppen utføres vekselvis, til testen gir false som verdi
- Eksempel:

```
int a = 1;  
while (a < 4) {  
    a += 2;  
}
```



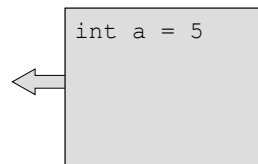
25

25

# while

- **while**-setningen endrer ikke arket, men spesifiserer regler for hvordan pilen flytter seg
  - testen og kroppen utføres vekselvis, til testen gir false som verdi
- Eksempel:

```
int a = 1;  
while (a < 4) {  
    a += 2;  
}
```



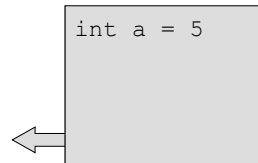
26

26

# while

- **while**-setningen endrer ikke arket, men spesifiserer regler for hvordan pilen flytter seg
  - testen og kroppen utføres vekselvis, til testen gir false som verdi
- Eksempel:

```
int a = 1;
while (a < 4) {
    a += 2;
}
```



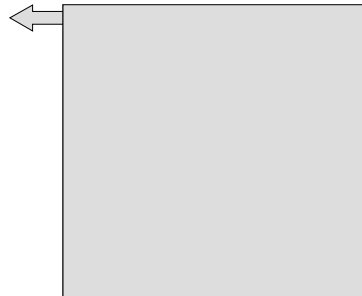
27

27

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen
- Eksempel:
 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```
- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken



28

28

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



```
int sum = 0
```

- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

29

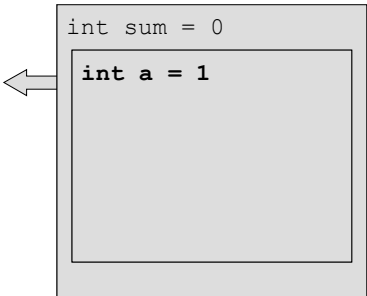
29

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



```
int sum = 0
```

```
int a = 1
```

- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

30

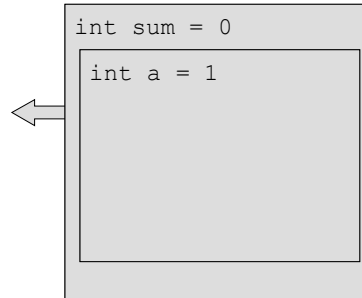
30

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

31

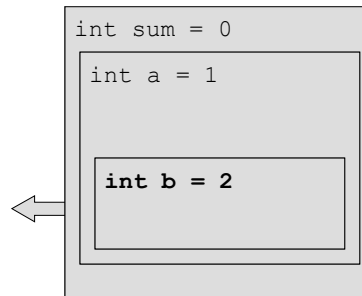
31

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

32

32

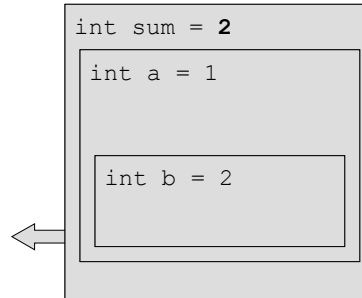


# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

33

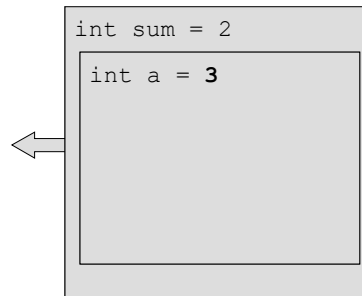
33

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

34

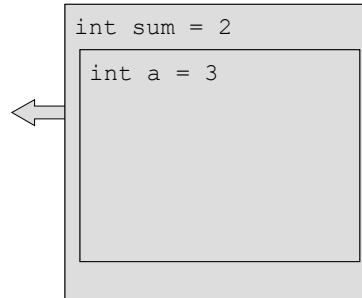
34

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

35

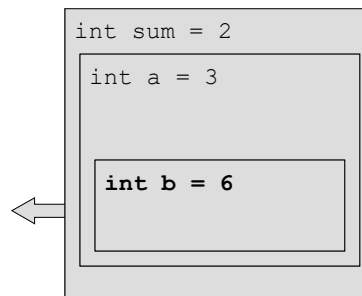
35

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

36

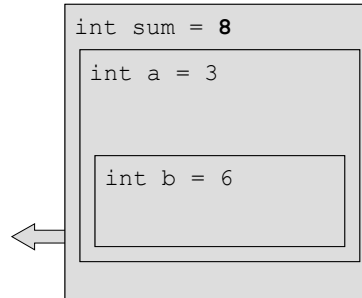
36

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

37

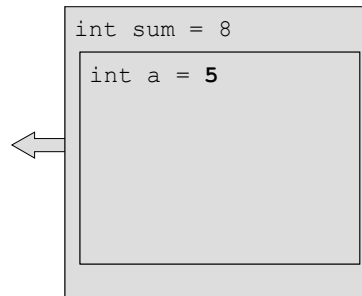
37

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

38

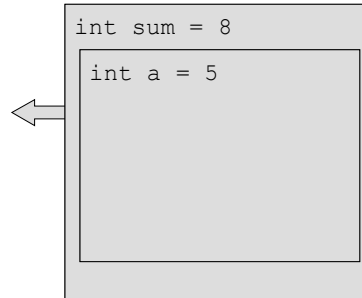
38

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

39

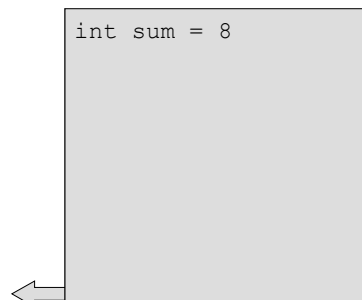
39

# for

- **for**-setningen har subtile forskjeller fra **while**, fordi den introduserer en implisitt { . . . }-blokk for init-, test- og steg-delen

- Eksempel: 

```
int sum = 0;
for (int a = 1;
    a < 4;
    a += 2)
{
    int b = a*2;
    sum += b;
}
```



- test- og steg-delen kan ikke referere til temporære variabler i den indre { . . . }-blokken

40

40

# Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```

- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

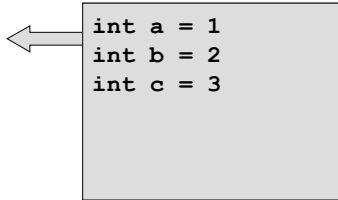
41

41

# Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



```
int a = 1
int b = 2
int c = 3
```

- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

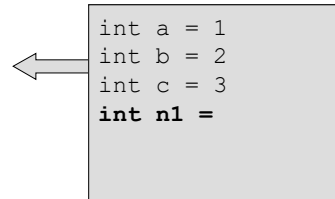
42

42

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

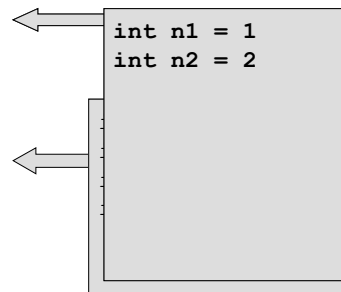
43

43

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

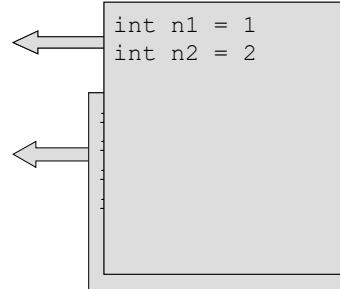
44

44

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

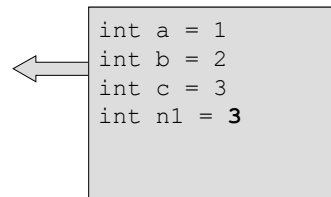
45

45

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene


46

46

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



```
int a = 1
int b = 2
int c = 3
int n1 = 3
int n2 =
```

- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene


47

47

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



```
int n1 = 3
int n2 = 3
```

- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

48

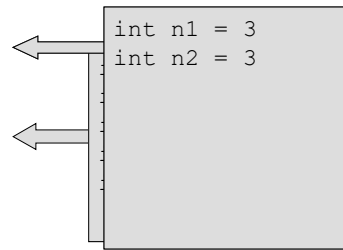
48



## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

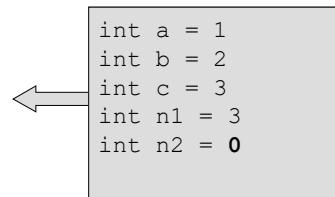
49

49

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

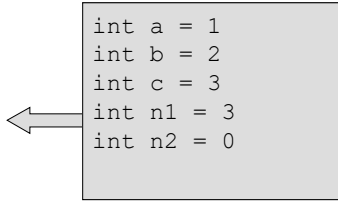
50

50

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3)
```



```
int a = 1
int b = 2
int c = 3
int n1 = 3
int n2 = 0
```

- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

51

51

## Egendefinerte funksjoner

- Når funksjoner kalles, legges et nytt ark på, som skygger for de andre. Argumentene brukes til å initialisere parametrene som variabler på det nye arket:

```
int x(int n1, int n2)
{return n1 + n2;}
int y(int n1, int n2)
{return n1 - n2;}
int z(int a, int b, int c)
{int n1 = x(a, b);
 int n2 = y(n1, c);
 return n2;}
z(1, 2, 3) => 0
```

- Variablene i den kallende funksjonen, er ikke tilgjengelig i den kalte funksjonen. De kan kun kommunisere gjennom parametrene/argumentene

52

52

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```

- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

53

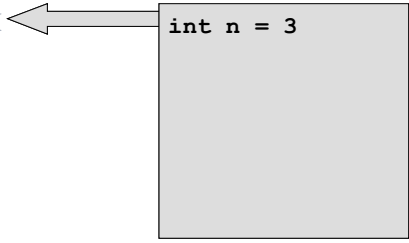
53

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
static int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
```

fak(3)



int n = 3

- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

54

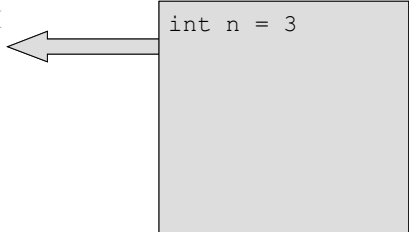
54

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
static int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```

int n = 3



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

55

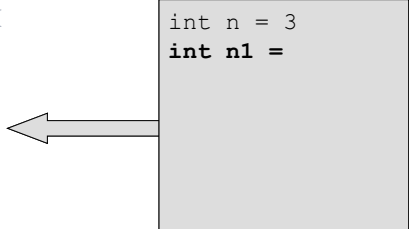
55

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
static int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```

int n = 3  
int n1 =



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

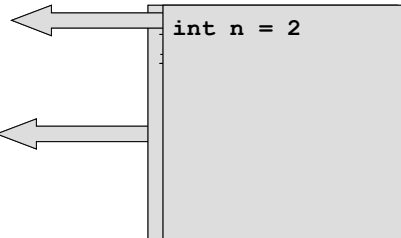
56

56

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
static int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

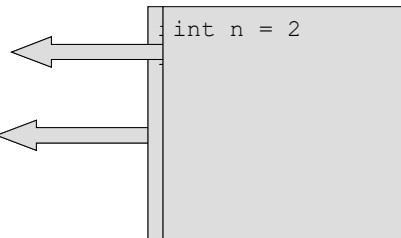
57

57

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

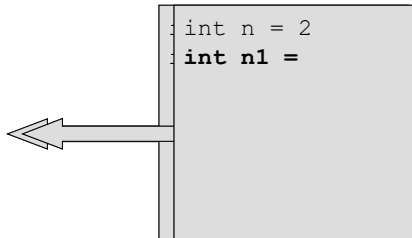
58

58

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

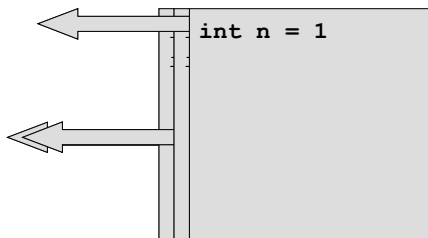
59

59

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

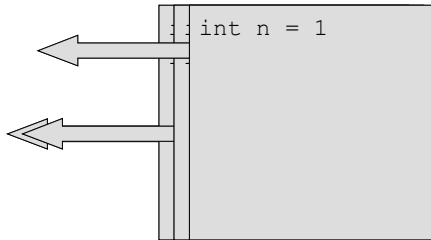
60

60

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

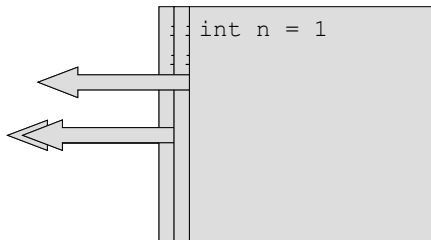
61

61

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

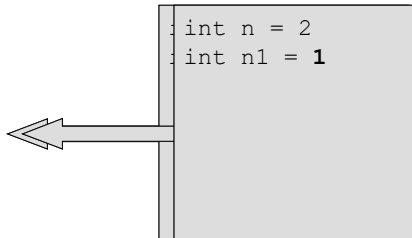
62

62

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

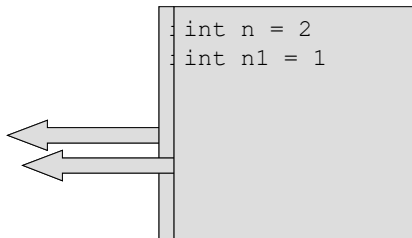
63

63

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

64

64

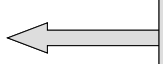


# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```

```
int n = 3
int n1 = 2
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

65


65

# Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3)
```

```
int n = 3
int n1 = 2
```



- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

66

66

## Rekursive funksjoner

- Samme mekanisme brukes når funksjoner kaller seg selv. Selv om slike funksjoner er vanskelige å skrive rett, er prinsippet det samme.

```
int fak(int n) {
    if (n <= 1) {
        return 1;
    }
    int n1 = fak(n-1);
    return n * n1;
}
fak(3) => 6
```

- Oppgave: Prøv å kalle følgende funksjon med 3 som argument:

```
int fib(int n)
{ return (n <= 1 ? n : fib(n-1) + fib(n-2)); }
```

67

67

## Objekter

- Denne prosedyreorienterte modellen må utvides til å håndtere objekter:
  - objekter fungerer som egne ark med variabler
  - hver metode utføres i tilknytning til et bestemt objekt
  - this**-konstanten er en referanse til dette objektet
  - this** er implisitt når det refereres direkte til variabler og metoder uten bruk av .-notasjonen
  - Alle metoder i et objekt vil kunne se this-variablene
- Merk at **static**-metoder og –variabler fungerer som vanlige prosedyreorienterte globale variable og metoder, uten noen objektkobling og **this**

68

68

# Metoder

- ```

public class Ansatt {
    public String navn;
    public int kontor;
    public String getNavn() {
        return navn;
    }
    public void setNavn(String s) {
        navn = s;
    }
}

```
- ```

public static void main(...) {
    Ansatt a = new Ansatt();
    a.setNavn("hal");
    // .-notasjonen og metodenavn
    if (a.getNavn() == "hal") {
        System.out.println("Ja, det virke
    }
}

```
- Arkmodellen må utvides ved at instansarket/kortet omslutter/ses av metodens ark

String navn =  
int kontor = 0

Ansatt a =

69

69

# Metoder

- ```

public class Ansatt {
    public String navn;
    public int kontor;
    public String getNavn() {
        return navn;
    }
    public void setNavn(String s) {
        navn = s;
    }
}

```
- ```

public static void main(...) {
    Ansatt a = new Ansatt();
    a.setNavn("hal");
    // .-notasjonen og metodenavn
    if (a.getNavn() == "hal") {
        System.out.println("Ja, det virke
    }
}

```
- Arkmodellen må utvides ved at instansarket/kortet omslutter/ses av metodens ark

String navn =  
int kontor = 0

Ansatt a =

70

70

## Metoder m/this-referanse

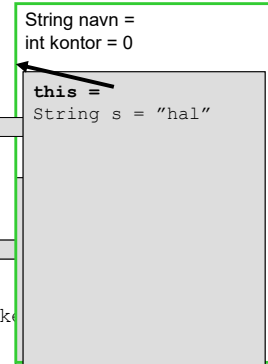
- ```

public class Ansatt {
    public String navn;
    public int kontor;
    public String getNavn() {
        return navn;
    }
    public void setNavn(String s) {
        navn = s;
    }
}

```
- ```

public static void main(...) {
    Ansatt a = new Ansatt();
    a.setNavn("hal");
    // .-notasjonen og metodenavn
    if (a.getNavn() == "hal") {
        System.out.println("Ja, det virke
    }
}

```
- Metodens ark har en spesiell *referanse* til instansen, i form av en konstant med navn **this**. Denne brukes for å nå objektet selv, samt objektets variabler og metoder.



71

71

## Metoder m/this-referanse

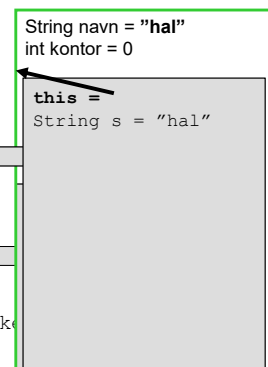
- ```

public class Ansatt {
    public String navn;
    public int kontor;
    public String getNavn() {
        return navn;
    }
    public void setNavn(String s) {
        navn = s;
    }
}

```
- ```

public static void main(...) {
    Ansatt a = new Ansatt();
    a.setNavn("hal");
    // .-notasjonen og metodenavn
    if (a.getNavn() == "hal") {
        System.out.println("Ja, det virke
    }
}

```
- Metodens ark har en spesiell *referanse* til instansen, i form av en konstant med navn **this**. Denne brukes for å nå objektet selv, samt objektets variabler og metoder.



72

72

## Metoder m/this-referanse

- ```

public class Ansatt {
    public String navn;
    public int kontor;
    public String getNavn() {
        return navn;
    }
    public void setNavn(String s) {
        navn = s;
    }
}

```
- ```

public static void main(...) {
    Ansatt a = new Ansatt();
    a.setNavn("hal");
    // .-notasjonen og metodenavn
    if (a.getNavn() == "hal") {
        System.out.println("Ja, det virker")
    }
}

```
- Metodens ark har en spesiell *referanse* til instansen, i form av en konstant med navn **this**. Denne brukes for å nå objektets selv, samt objektets variabler og metoder.

String navn = "hal"  
int kontor = 0

Ansatt a =

73

73

## Prøv selv

```

public class P {
    P p;
    P g() {
        return p;
    }
    void z(P p) {
        if (p == this.p) {
            return;
        }
        P oldP = this.p;
        this.p = p;
        if (oldP != null && oldP.g() == this) {
            oldP.z(null);
        }
        if (this.p != null) {
            this.p.z(this);
        }
    }
}

```

main:

```

P p1 = new P();
P p2 = new P();
p1.z(p2);

P p3 = new P();
P p4 = new P();
p3.z(p4);

p1.z(p4);

```

74

74

# Globale variabler

- Globale variabler lever like lenge som hele programmet
  - de dukker opp når programmet starter
  - de ”lever” så lenge programmet er aktivt
  - de forsvinner først når programmet avslutter
- Globale variabler ligger på et kjempeark under alle andre ark, også de knyttet til funksjonskall
  - kan endres av all kode, også av koden inni egendefinerte funksjoner
  - gjør det mulig for funksjoner å formidle resultater til andre deler av programmet, ut over return-verdien
- Globale variabler bør brukes med forsiktighet, fordi det gjør det vanskeligere å isolere programdeler fra hverandre

75

75

# Globale variabler

## • Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tabell) {
    for (int i = 0; i < tabell.length; i++) {
      int n = Integer.valueOf(tabell[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```

```

int sum = 0
int kvadratSum = 0

```

- Hva skrives ut dersom programmet kalles med ”1”, ”2” og ”3” som kommandolinjeargumenter?

76

76

# Globale variabler

- Eksempel:

```
- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tabell) {
    for (int i = 0; i < tabell.length; i++) {
      int n = Integer.valueOf(tabell[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };
```

```
int sum = 0
int kvadratSum = 0
```

```
String[] args =
{"1", "2", "3"}
```

- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

77

77

# Globale variabler

- Eksempel:

```
- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tabell) {
    for (int i = 0; i < tabell.length; i++) {
      int n = Integer.valueOf(tabell[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };
```

```
int sum = 0
int kvadratSum = 0
```

```
String[] args =
{"1", "2", "3"}
```

- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

78

78

# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```

```

int sum = 0
int kvadratSum = 0

String[] tab =
{"1", "2", "3"}

```

- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

79

79

# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```

```

int sum = 0
int kvadratSum = 0

String[] tab =
{"1", "2", "3"}

int i = 0

```

- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

80

80



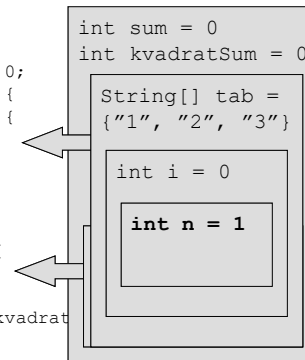
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

81

81

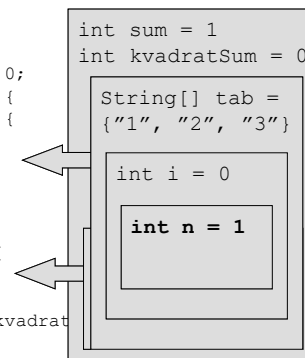
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

82

82

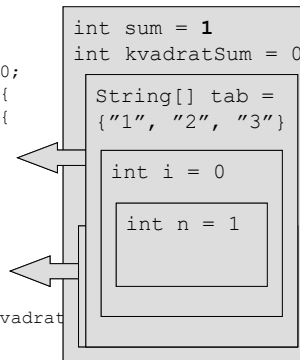
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

83

83

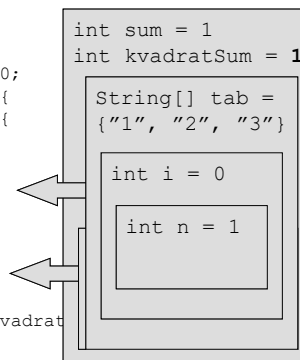
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

84

84

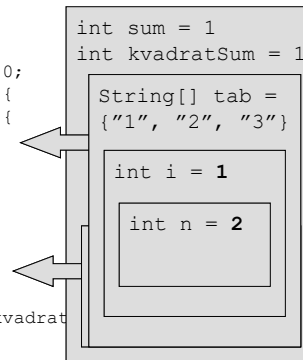
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

85

85

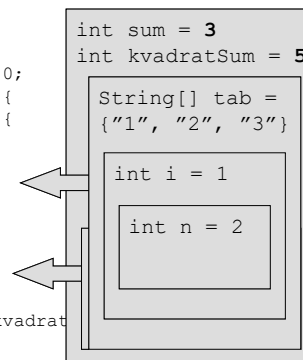
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

86

86

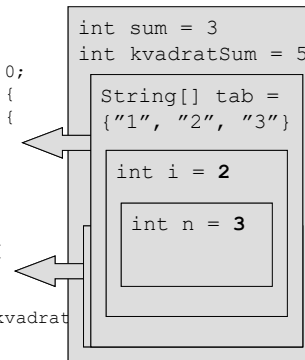
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

87

87

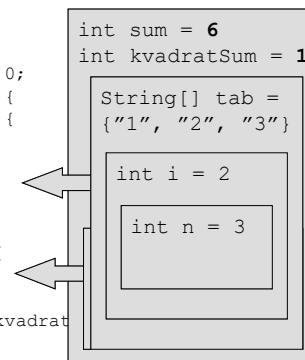
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

88

88

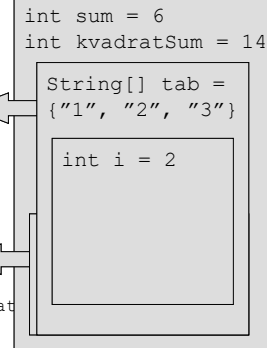
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

89

89

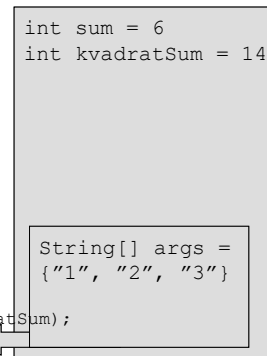
# Globale variabler

- Eksempel:

```

- public static int sum = 0, kvadratSum = 0;
  public static void summer(String[] tab) {
    for (int i = 0; i < tab.length; i++) {
      int n = Integer.valueOf(tab[i]);
      sum += n;
      kvadratSum += n * n;
    }
  }
  public static void main(String[] args) {
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
    summer(args);
    System.out.println("Sum: " + sum);
    System.out.println("Kvadratsum: " + kvadratSum);
  };

```



- Hva skrives ut dersom programmet kalles med "1", "2" og "3" som kommandolinjeargumenter?

90

90