NTNU

Kunnskap for en bedre verden

IDAT1001 Programmering 1

Kontrollstrukturen løkke

Mål for dagen:

Bli kjent med kontrollstrukturen løkke

Kunne illustrere løkke ved hjelp av aktivitetsdiagram

Kunne programmere løkker ved hjelp av while, for og do-while

Kunne nøste valg og løkkestrukturer med hverandre

Testing av løkker

En løkke med en sammensatt betingelse

Muhammad Ali Norozi

Oppmøteregistrering for smittesporing

- Scan QR-kode med egen mobil
- Logg på med FEIDE-bruker
- Registrer tidsrom og bordnummer
- https://innsida.ntnu.no/checkin/room/38377

Læringsutbytter, forelesning 2

- Kunne identifisere kontrollstrukturen valg og illustrere den ved hjelp av aktivitetsdiagram
- Programmere valg ved hjelp av if-setning og flervalg ved hjelp av nøstet if-setning og ved bruk av switch
- Bruke betingelsesoperatoren ?:
- Bruke logiske variabler og enkle sammensatte logiske uttrykk
- Lage strukturert og oversiktlig programkode ved bruk av innrykk og navnekonvensjon
- Kunne gjøre nøyaktige beregninger ved regning med desimaltall slik at avrundingsfeil unngås.

Agenda

- Repetisjon fra forrige uke
  - Valg, aktivitetsdiagram, if-elseif-else, switch, blokker, logiske uttrykk
- Læringsutbytter kapittel 3.8
- Løkke, aktivitetsdiagram, while-setningen og operatorer
- En løkke med generell betingelse, for-løkker og do-while
- Valg av riktig løkkesetning, testing av løkker, nøstede kontrollstrukturer, break og continue
- **Innledende oppgaver - jobb gjerne sammen**

Læringsutbytter, forelesning 3

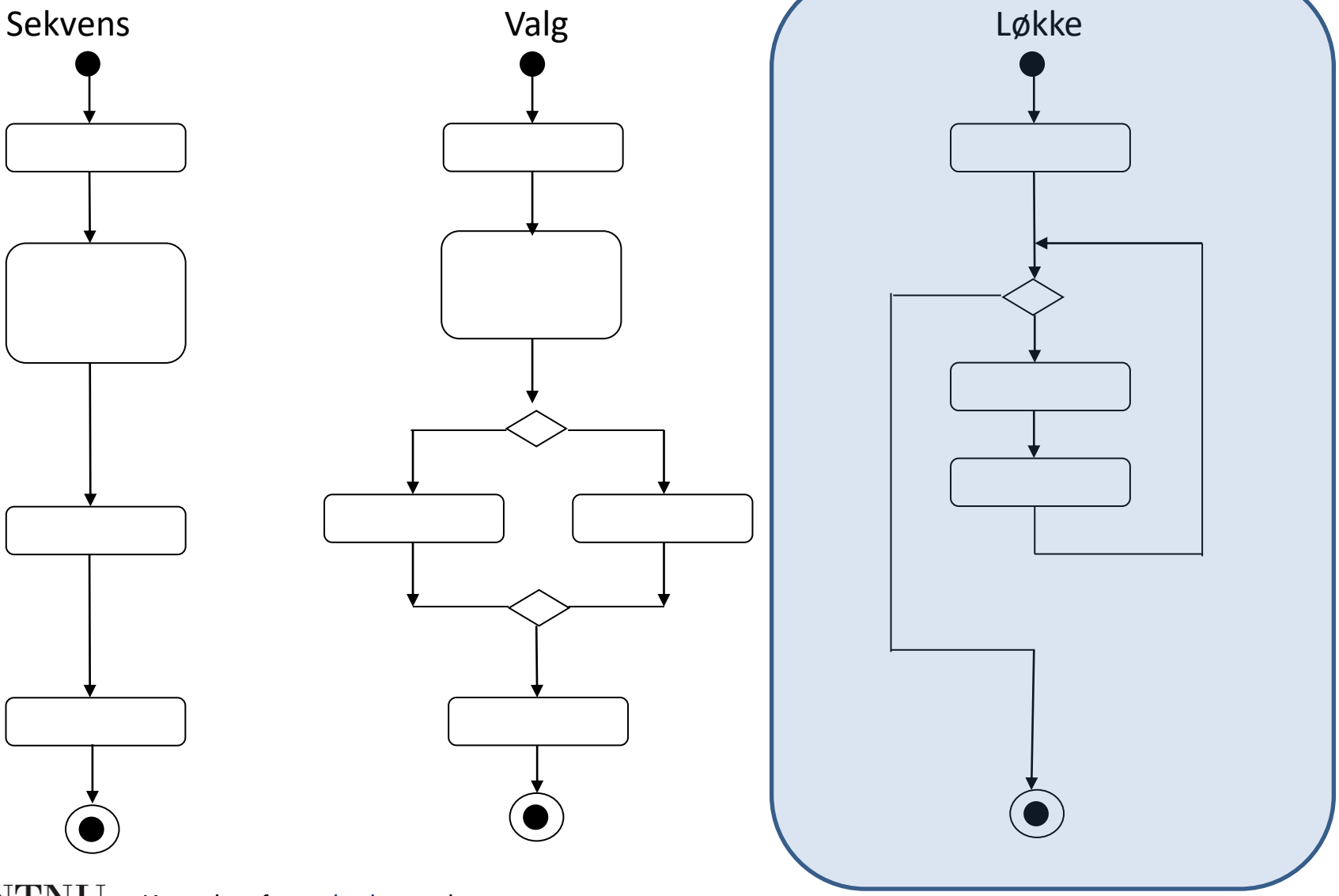
- Identifisere kontrollstrukturen løkke i en problembeskrivelse
- Illustrere kontrollstrukturen løkke ved hjelp av aktivitetsdiagram
- Identifisere de ulike delene av kontrollstrukturen: initiering, oppdatering, løkkebetingelse, løkkekropp
- Anvende operatorene ++, - -, +=, -=, *=, /= og %=
- Anvende riktig setning (for-, while eller do-setningen) for å programmere en løkke
- Sette opp testdata for å validere programkoden for løkke
- Programmere nøstede kontrollstrukturer (løkker og valg)
- Lese programmer der break og continue er brukt

Løkke som kontrollstruktur

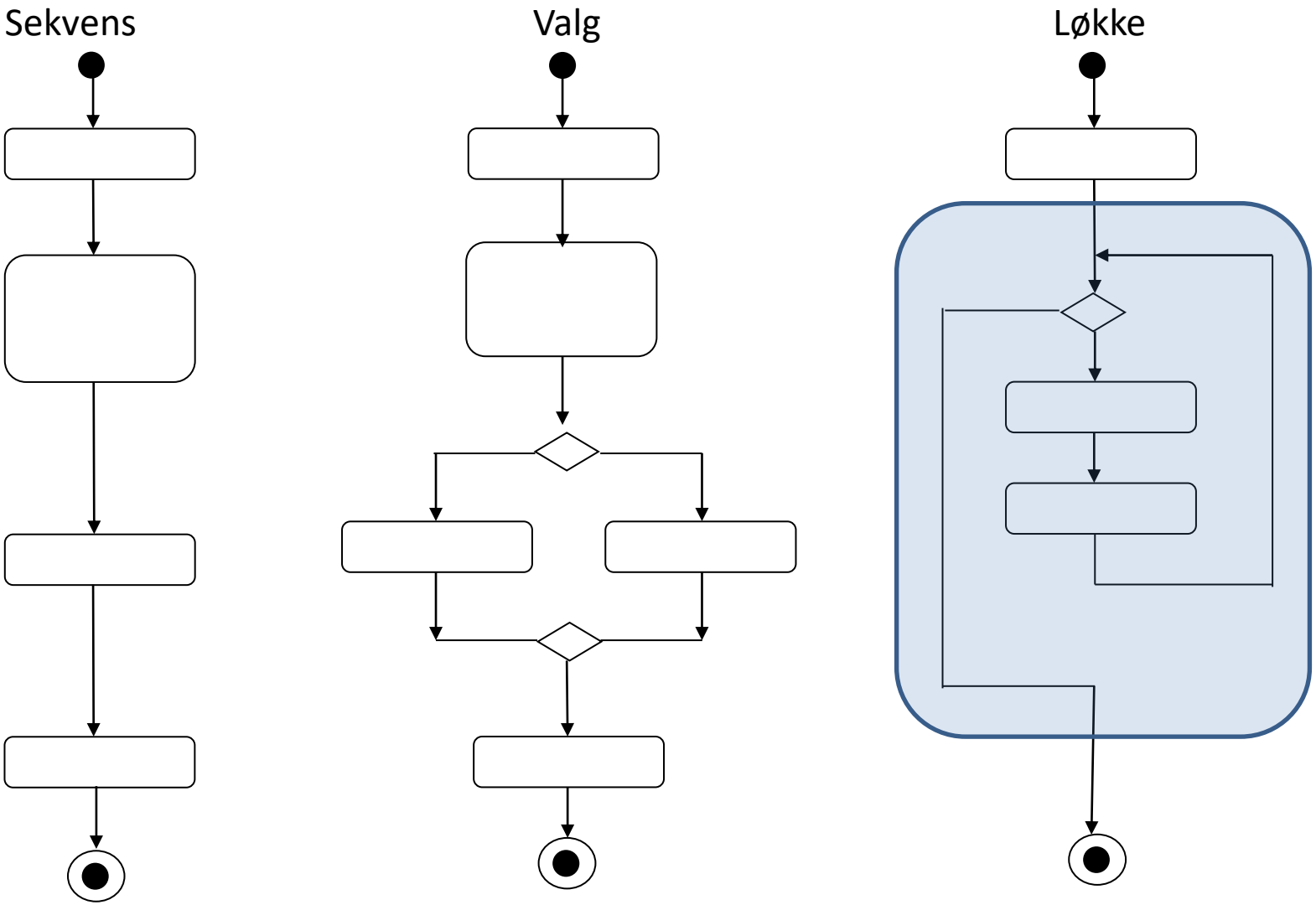
- En algoritme er et begrenset og ordnet sett med veldefinerte regler for løsning av et problem.
- Tre kategorier rekkefølge / kontrollstrukturer:
  - sekvens
  - valg
  - løkke

- En løkke gjør det mulig å gjenta en enkelt setning eller en blokk én eller flere ganger.
- En tellerkontrollert løkke/ Betingelsesstyrt løkke – SPM: er antall gjennomløp kjent på forhånd eller ikke.

UML - AKTIVITETSDIAGRAM



UML - AKTIVITETSDIAGRAM



Ulike løkketyper

- while-løkke
- for-løkke
- do-while-løkke

- I tillegg finnes det en såkalt utvidet for-løkke (for each) - mer om denne senere i kurset.

while-setningen med eksempel

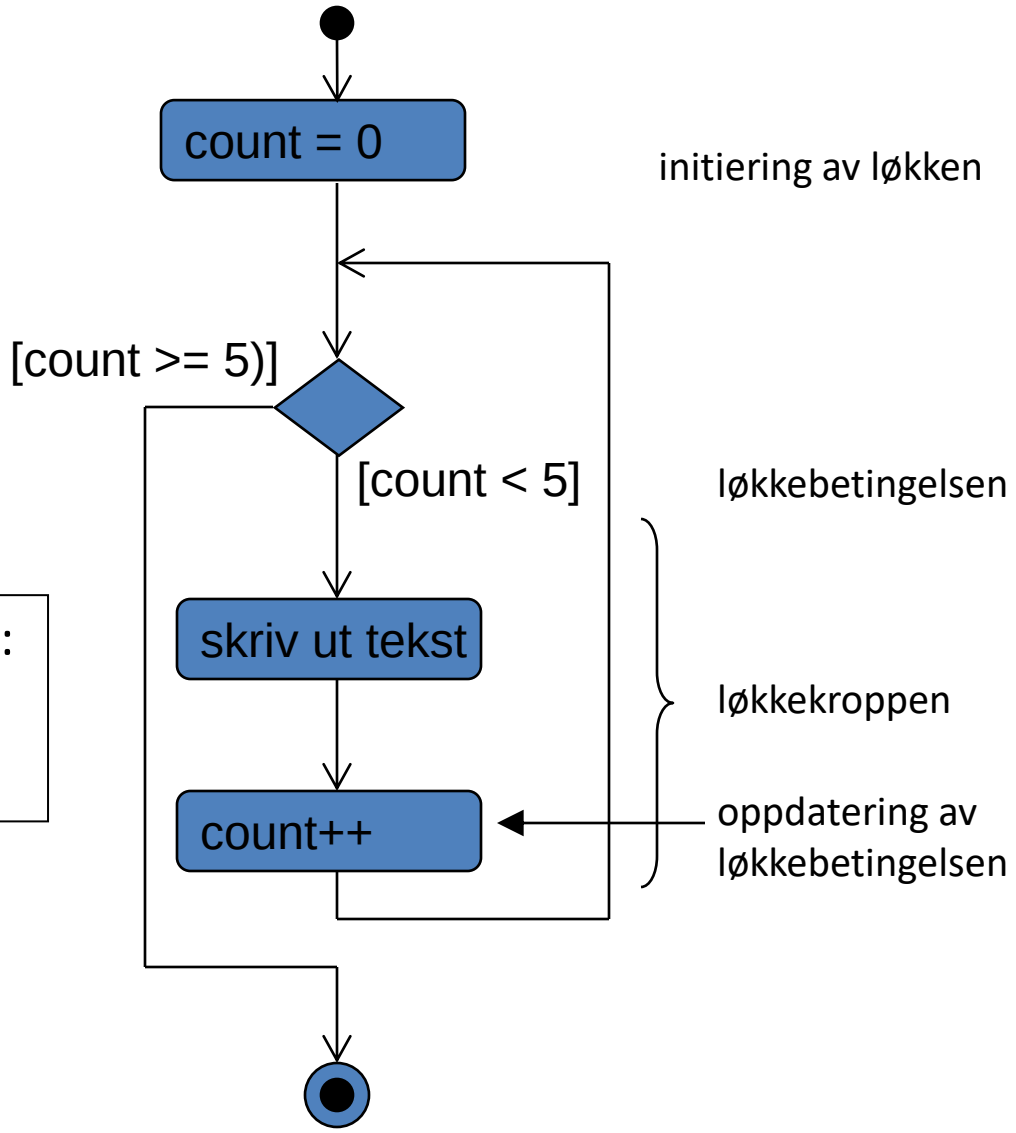
```
class PrintSeveralLines {
  public static void main(String[] args) {
    int count = 0;
    while (count < 5) {
      System.out.println("Dette er en linje");
      count++;
    }
  }
}
```

Utskrift:

```
Dette er en linje
Dette er en linje
Dette er en linje
Dette er en linje
Dette er en linje
```

while-setningen:

while (betingelse)
  setning/blokk



Flere operatorer for de fire regneartene

- Kjenner fra før: +, -, *, /, %

- Inkrementoperatoren ++. antall++
  øker variabelen antall med 1
- Dekrementoperatoren --. antall--
  reduserer verdien med 1
- Sammensatt tilordningsoperator. antall += 5;
  tilsvarer antall = antall +5;

- Husk prioritet når du utfører beregningene

Oppgave operatorer

Oppgave 1

- Hva blir skrevet ut?

```
int tall = 2;
tall++;
tall--;
System.out.println(tall);
tall+=5;
tall*=2;
tall%=3;
System.out.println(tall);
```

2

Oppgave 2

- Hva inneholder a, b og c?

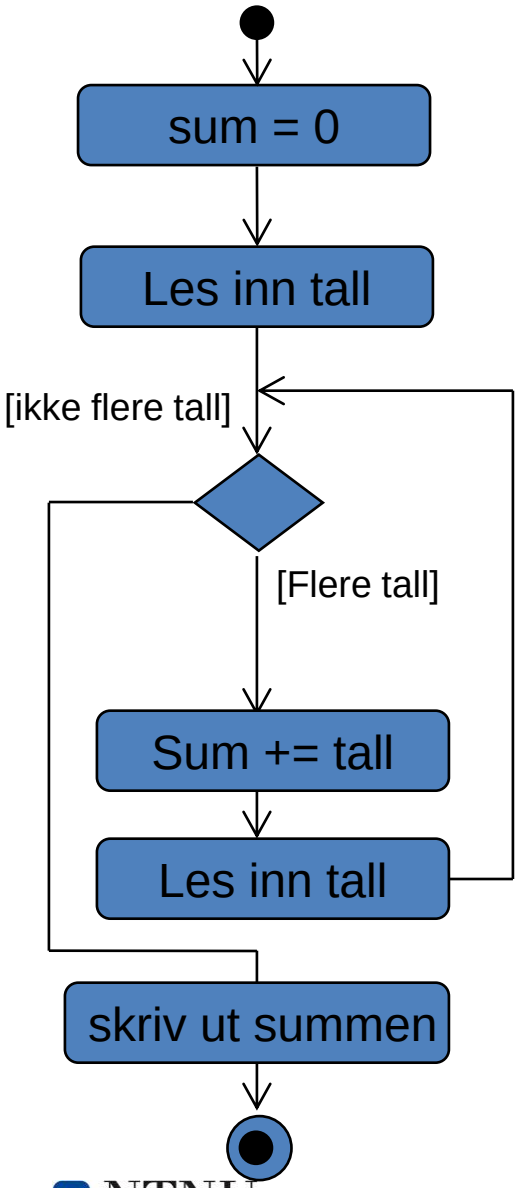
```
int a = 10;
int b = 20;
int c = b % a;
a++;
b++;
c += a;
a *= c;
b--;
c /= 2;
c %= 2;
```

a = 121

b = 20

c = 1

En løkke med en generell betingelse



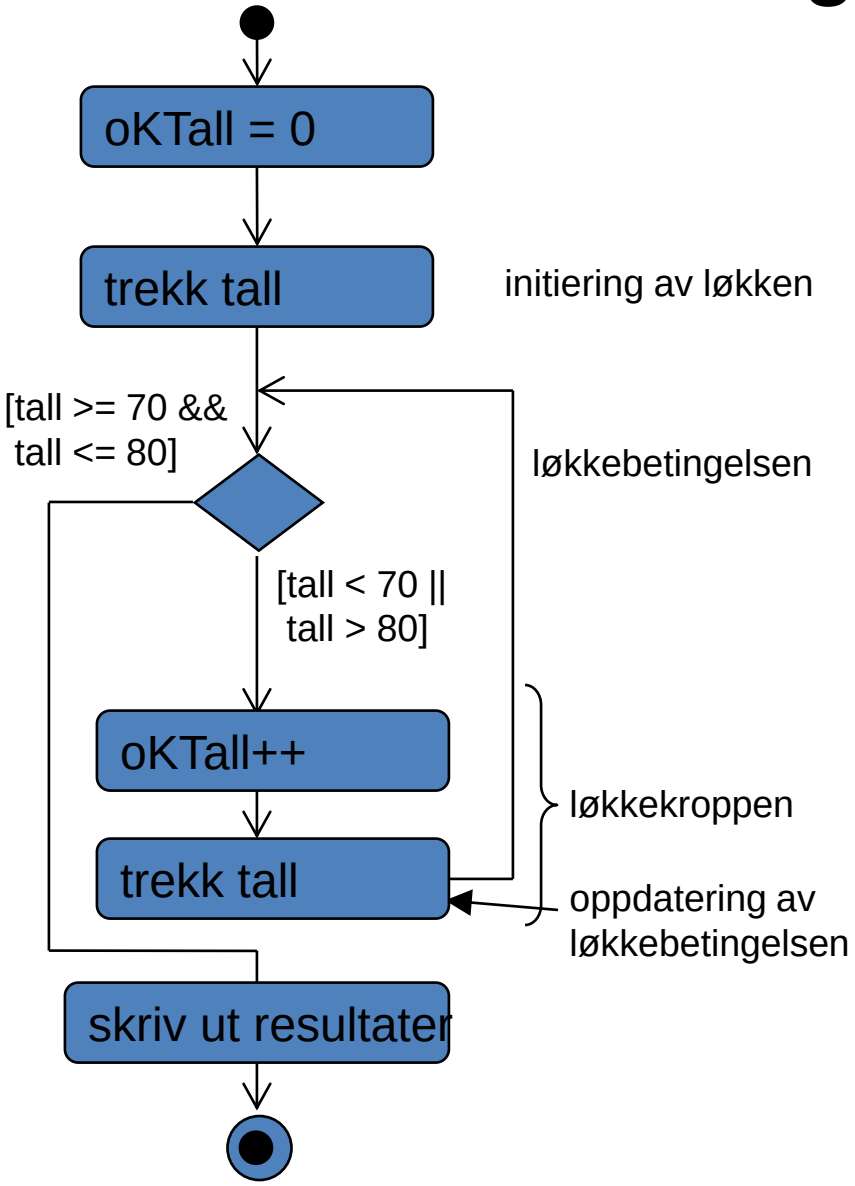

```
import static javax.swing.JOptionPane.*;
class Sum {
  public static void main(String[] args) {

    int sum = 0;
    String numberAsText = showInputDialog (”Skriv tall, avslutt
med Esc”);

    while (numberAsText != null) {
        int number = Integer.parseInt(numberAsText);
        sum += number;
        numberAsText = showInputDialog (”Skriv tall, avslutt med Esc”);
    }

    showMessageDialog(null, ”Summen er ” + sum+”.”);
  }
}
```

En løkke med en generell betingelse




```
class FindNumber {
  public static void main(String[] args) {
    final int LIMIT1 = 0;
    final int limit2 = 70;
    final int limit3 = 80;
    final int limit4 = 100;

    Random numberGen = new Random();
    int oKNumber = 0;
    int tall =limit1 + numberGen.nextInt(limit4 - limit1 + 1);

    while (tall < limit2 || tall > limit3) {
      oKNumber++;
      System.out.println(number);
      tall = limit1 +
            numberGen.nextInt(limit4 - limit1 + 1);
    }
    System.out.println("Vi måtte trekke " + oKNumber +
                       " tall til vi fant et ugyldig: " + number);
  }
}
```

Å huske en dataverdi fra ett løkke-gjennomløp til neste

```
Import static javax.swing.JOptionPane.*;
class SumUtenLikeTall {
  public static void main(String[] args) {
    int antallTallSummert = 0;
    int sum = 0;
    String forrigeTallLest = ””;
    String tallLest = showInputDialog (”Skriv ett tall av gangen, avslutt med Esc: ”);

     while (tallLest != null) {
       if (!tallLest.equals(forrigeTallLest)){
         int tall = Integer.parseInt(tallLest);
         sum += tall;
         antallTallSummert++;
       }
       forrigeTallLest = tallLest;
       tallLest = showInputDialog (”Skriv ett tall av gangen, avslutt med Esc”);
     }
    ShowMessageDialog(null, ”Summen er ” + sum +”. Vi har summert ” + antallTallSummert + ”
tall.”);
   }
 }
```

Løkke med for-setningen

```
int count = 0;
    while (count < 5) {
      System.out.println("En linje");
      count++;
    }
```

```
for (int count = 0; count < 5; count++) {
      System.out.println("En linje");
    }
```

- Tellerkontrollerte løkke, vi vet antall løkkegjennomløp: Passende med for-løkke
- Oppgave: Identifiser initiering, betingelse og oppdatering av løkkebetingelsen i de to løkkekonstruksjonene over.
- Definisjonsområdet til variabler deklarert i for-setningen er for-setningen selv.
- Syntaks – for-løkke

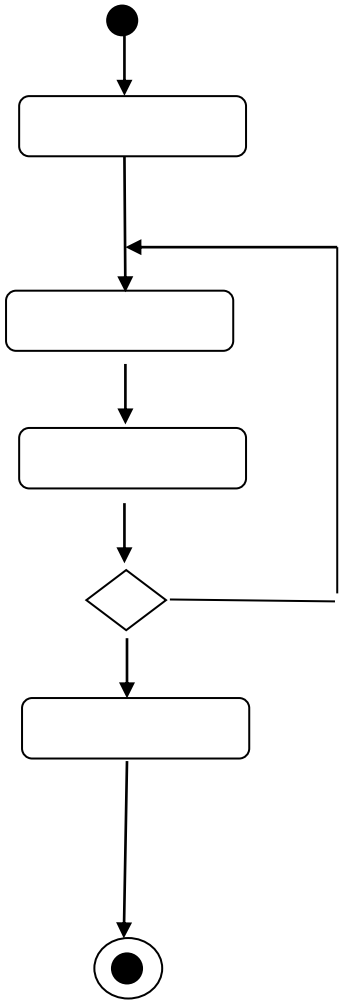
  for (initiering; betingelse; oppdatering av betingelse)

  Setning/ blokk

Løkker med do-while-setningen

- do-while-setningen skiller seg fra for- og while-setningene ved at løkkekroppen **alltid** gjennomløpes minst en gang
- Praktisk ved innlesing der data skal kontrolleres
- Eksempel:

```
int antStudenter;
do {
    String lestTekst = showInputDialog("Oppgi antall studenter
    (maks. 30): ");
    antStudenter = Integer.parseInt(lestTekst);
} while (antStudenter > 0 || antStudenter <= 30);
```



- Syntaks:

  do

  setning/blokk

  while (logisk uttrykk)

Valg av riktig løkke-setning

- Bruk **for-setningen** dersom antall gjennomløp av løkken er kjent på forhånd, eller der en heltallsvariabel øker eller avtar med en fast verdi for hvert gjennomløp.

- Bruk **while-setningen** dersom antall løkkegjennomløp styres av en generell betingelse.

- Bruk **do-while-setningen** dersom antall løkkegjennomløp styres av en generell betingelse, og løkken alltid skal gjennomløpes minst en gang.

Testing av løkker

- Formuler testdata som fanger opp følgende tilfeller:
  - Løkken gjennomløpes eksakt 1 gang
  - Løkken gjennomløpes noen ganger
  - Løkken gjennomløpes 0 ganger (unntak for do-setninger)
  - Hvis betingelsen er sammensatt må de ulike delene testes hver for seg

  - Er det mulig å konstruere testdata som gjør at løkkebetingelsen aldri blir usann? Hvis ja, vil dette kunne bli en evig løkke, og det var vel ikke meningen?

Nøstede kontrollstrukturer

- if-setninger inne i hverandre, løkker inne i hverandre, if-setninger inne i løkker og omvendt, osv...

- Hva skrives ut når følgende programkode kjører?

```
for (int linjeteller = 0; linjeteller < 10; linjeteller++) {
   int sum = 0;
    for (int tall = 1; tall <= linjeteller; tall++) {
          sum += tall;
          System.out.print(tall + " ");
     } // for tall
     System.out.println("Summen blir: " + sum);
  } // for linjeteller
```

```
Summen blir: 0
1 Summen blir: 1
1 2 Summen blir: 3
1 2 3 Summen blir: 6
1 2 3 4 Summen blir: 10
1 2 3 4 5 Summen blir: 15
1 2 3 4 5 6 Summen blir: 21
1 2 3 4 5 6 7 Summen blir: 28
1 2 3 4 5 6 7 8 Summen blir: 36
1 2 3 4 5 6 7 8 9 Summen blir: 45
```

En løkke med en sammensatt betingelse

- while (tallLest != null && !grenseNådd)

```
Import static javax.swing.JOptionPane.*;
class SumWithLimit {
  public static void main(String[] args) {

   final int LIMIT = 10;
   boolean limitReached = false;
   int sum = 0;
    String numberAsText = showInputDialog (”Skriv tall, avslutt med Esc”);

    while (numberAsText != null && !limitReached) {
              int number = Integer,parseInt(numberAsText);
              sum += number;
              if (sum > LIMIT){
                 limitReached = true;
               } else {
                 numberAsText = showInputDialog (”Skriv tall, avslutt med Esc”);
              }
   }

    ShowMessageDialog(null, ”Summen er ” + sum+”.”);
   }
}
```

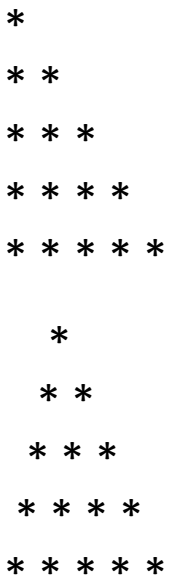
break og continue

- Lite brukt og bør i utgangspunktet unngås
  - Bruk heller if-else struktur
  - Uoversiktlig kode (vanskelig å lese)
- break medfører direkte uthopp fra den aktuelle setningen
  - Hensiktsmessig i forbindelse med switch
- continue medfører at resten av løkkekroppen hoppes over og programkontrollen fortsetter med neste gjennomløp
- Programeksempel:
  - `SumWithLimits_UsingBreak.java` og `SumEvenNumbersOnly_UsingContinue.java`

Innledende øvingsoppgave

Oppgave A
Lag et program som tegner følgende figur på skjermen

Brukeren skal selv bestemme antall linjer. Programmet skal avsluttes dersom brukeren skriver inn 0.



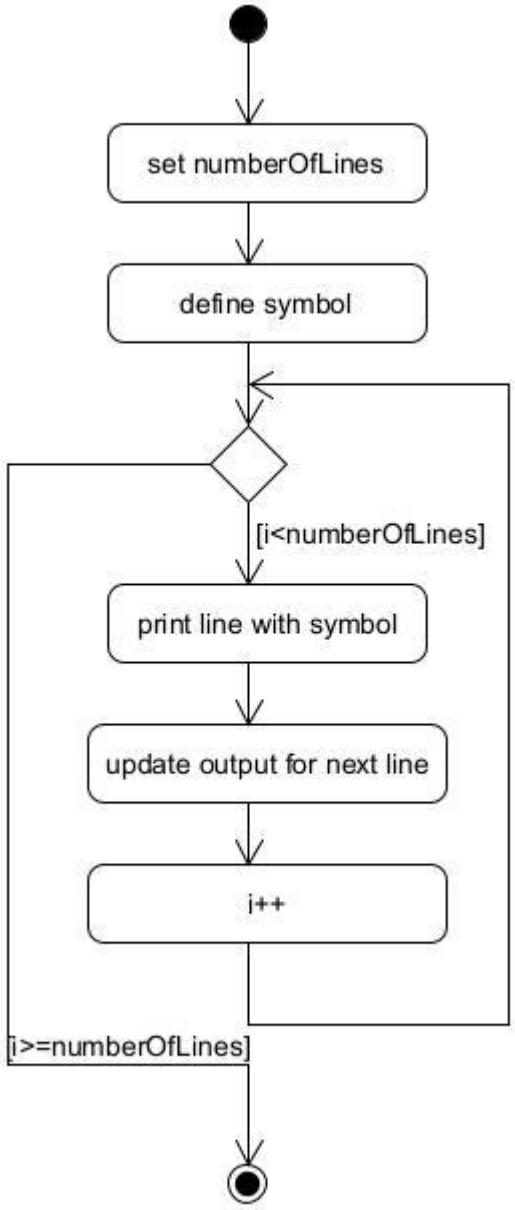
Oppgave B
Modifiser programmet slik at det skriver ut en likebeint trekant

- Sett opp et aktivitetsdiagram (før du begynner på java-koden)
- Programmer løsningen i henhold til ditt aktivitetsdiagram
- Sett opp testdatasett og test programmet

Likebeint trekant

- Antall linjer = 5
- Symbol = *
- Ønsket utskrift:

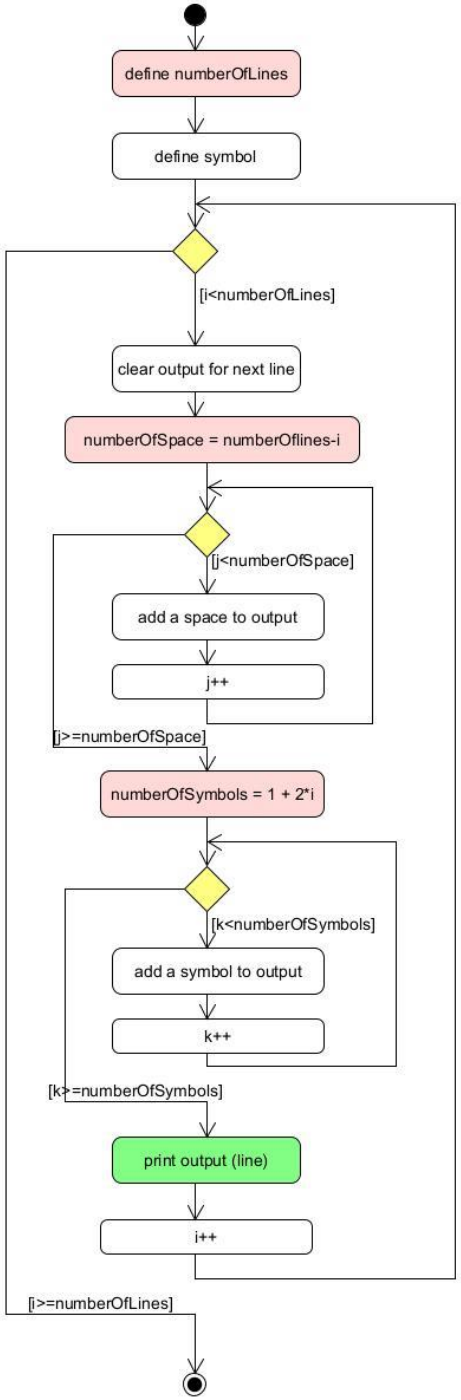
i					
0	*				
1	*	*			
2	*	*	*		
3	*	*	*	*	
4	*	*	*	*	*



Likesidet trekant

- NumberOfLines= 5
- Symbol = *
- numberOfSpace = numberOfLines – i
- numberOfSymbols = 1 + 2*i

i										
0						*				
1					*	*	*			
2				*	*	*	*	*		
3			*	*	*	*	*	*	*	
4		*	*	*	*	*	*	*	*	*



Caseoppgave – oppgavebeskrivelse

Oppgave A – Alle bord nærmest vinduene i programmeringslabben

Du har kjøpt et hjemmekinoanlegg som koster 17 600 kroner inkludert alle avgifter. Du skal betale for stereosystemet etter en avbetalingsplan med følgende betingelser:

Serielån, ingen betaling ved kjøpsdato, 7.5 % rente, månedlige avdrag på 550 kr

Lag et program som skriver ut en nedbetalingsplan som viser hva du faktisk må betale, hvor mye som er renter, hvor mye som er avdrag og ny saldo hver måned.

Oppgave B – Alle bord nærmest glassveggen i programmeringslabben

Du har kjøpt et stereoanlegg som koster 9 681,50 kroner inkludert alle avgifter. Du skal betale for stereoanlegget etter en avbetalingsplan med følgende betingelser:

Annuitetslån, ingen betaling ved kjøpsdato, 7.5 % rente, månedlige innbetaling på 550 kr.

Lag et program som skriver ut en nedbetalingsplan som viser hva du faktisk må betale, hvor mye som er renter, hvor mye som er avdrag og ny saldo hver måned.

- Sett opp et aktivitetsdiagram for din oppgave (før du begynner på java-koden)
- Programmer løsningen i henhold til ditt aktivitetsdiagram
- Sett opp testdatasett og test programmet