Løkker

En løkke er et sett med instruksjoner/funksjoner som blir kjørt om igjen mens en betingelse er sann. Java har 3 typer løkker som har samme grunnleggende funksjonalitet, men som er forskjellige i syntaks og hvor i løkken betingelsen blir sjekket. Alle betingelser må være uttrykk som gir en *booleansk* verdi. F.eks. `(10 < 7)` som blir *false* eller `(10 != 0 && 10>5)` som blir *true*.

while-løkke

En *while-løkke* tillater oss å kjøre kode om igjen basert på en booleansk betingelse. Det går an å tenke på en *while-løkke* som gjentatte *if*-utsagn.

Syntaks:

```
while(betingelse) {
    // Gjør noe
}
```

while-løkker starter med å evaluere betingelsen som er gitt. Hvis betingelsen er sann, blir koden i blokken kjørt. Normalt sett inneholder blokken kode som oppdaterer verdier som er relevant for betingelsen til neste gjennomgang. Når koden er kjørt, blir betingelsen evaluert på nytt, og fortsetter slik helt til betingelsen blir evaluert til *false*. **Eksempel**:

```
void loop() {
    int x = 1;

    while(x <= 4) {
        System.out.println(x);
        x++;
    }
}
```

for-løkke

En *for-løkke* gir oss en mer nøyaktig måte å definere løkken på. I motsetning til en *while-løkke* kan en *for-løkke* både initialisere en variabel, teste en betingelse, og øke/minste verdien av variabelen på en enkelt linje.

Syntaks:

```
for(initialisering teller; betingelse; minske/øke teller) {
    // Gjør noe
}
```

En *for-løkke* starter med å initialisere en variabel som fungerer som en teller. Vi kan bruke en allerede deklarert variabel, eller deklarere en ny variabel som befinner seg i *scopet* til løkken. Deretter evaluerer vi en betingelse som må returnere en *booleansk* verdi (true/false). Denne betingelsen blir sjekket før koden i løkken blir kjørt. Hvis betingelsen er *true*, blir koden kjørt. Når koden har kjørt, øker eller minsker vi verdien av telleren. Når betingelsen i løkken blir *false*, blir løkken avsluttet. **Eksempel**:

```
void loop() {
    for(int i = 0; i < 100; i+=10) {
        System.out.println(i);
    }
    // 0102030405060708090
}
```

Her har vi definert at *i* skal øke med 10 etter hver gjennomgang av løkken. Når *i* når 100, vil betingelsen bli *false* da 100 ikke er mindre enn 100, og løkken blir avsluttet.

Enhanced for-løkke

En *enhanced for-løkke* er en en type *for-løkke* som kan gjøre det enklere å gå gjennom tabeller og andre mengder. Denne løkken er lite fleksibel, og gir oss ikke indeksen til elementene vi går gjennom, og er derfor ikke alltid like godt egnet som en vanlig *for-løkke*.

Syntaks:

```
for(T element: mengde) { // Hvor T er datatypen
    // Gjør noe
}
```

La oss ta et eksempel som demonstrerer hvordan en *enhanced for-løkke* kan gjøre livet lettere. Si at vi har en tabell med navn, og at vi vil skrive ut alle disse navnene. Vi kan løse dette med både *for-* og *enhanced for-*løkker:

```
void print() {
    String arr[] = new String[]{"Ole", "Dole", "Doffen"};

    for(String x : arr) {
        System.out.println(x);
    }

    for(int i = 0; i < arr.length; i++) {
        System.out.println(arr[i]);
    }
}
```

I dette tilfellet bil det være enklere og mer lesbart å bruke en *enchanced for-løkke*, da vi ikke trenger indeksen til elementet.

do-while-løkke

En *do-while-løkke* ligner på en *while-løkke* hvor den eneste forskjellen er at *do-while-løkken* sjekker betingelsen først etter koden i blokken blir utført en gang.

Syntaks:

```
do {
    // Gjør noe
}
while(betingelse);
```

Her starter løkken med å først kjøre koden i blokken, før den evaluerer betingelsen. Hvis betingelsen evalueres til *true*, kjøres løkken en gang til. Hvis *false* avsluttes løkken.

Lett å gjøre feil

En av feilene man ofte kan gjøre når man implementerer en løkke, er å sette opp løkken slik at den kjøres uendelig, og at programmet ikke kommer seg ut av løkken. Dette skjer når betingelsen er feil av en eller annen grunn.

Vurder følgende kode:

```
void loop() {

    // Her vil betingelsen (i != 0) aldri bli false.
    for(int i = 5; i != 0; i -= 2) {
        System.out.print(i);
    }

    // Her oppdaterer vi ikke x i løkken.
    int x = 1;
    while(x == 1) {
        System.out.print("Hjelp, jeg sitter fast!");
    }
}
```

Begge disse løkkene vil kjøre uendelig, da betingelsen aldri vil bli *false*. I det første tilfellet er dette grunnet en dårlig betingelse, og i andre tilfellet er det grunnet `x` aldri blir endret, slik at betingelsen alltid vil være *true*.