Algoritmer og datastrukturer
E – Løkker i Java

E Løkker i Java

E.1 For-løkker

En for-løkke består av de fire delene *initialisering*, *betingelse*, *oppdatering* og *kropp* (eng: body). Et typisk eksempel er en for-løkke som traverserer en tabell *a*:

```
for (int i = 0; i < a.length; i++)
{
  // kroppen
}
```

Programkode* E.1 *a)

Her er initialiseringen gitt ved `int i = 0`, betingelsen er `i < a.length` og oppdateringen `i++`. Det som står i blokken innrammet av krøllparentesene { }, er for-løkkens kropp. Den består normalt av en eller flere programsetninger.

En generell for-løkke ser slik ut:

```
for (<initialisering>; <betingelse>; <oppdatering>)
{
  // kroppen
}
```

Programkode* E.1 *b)

I flg. eksempel brukes en for-løkke til å travsere en generisk liste ved hjelp av en *iterator*. La f.eks. listens generiske parametertype være *String*:

```
for (Iterator<String> i = liste.iterator(); i.hasNext(); )
{
  String s = i.next();

  // øvrige programsetninger
}
```

Programkode* E.1 *c)

Hvis kroppen har kun én setning, trengs ingen blokk, dvs. innramming av krøllparenteser:

```
for (int i = 0; i < a.length; i++) <programsetning>;
```

Det er lurt å ha en blokk selv med kun én setning. Da er blokken klar hvis det senere trengs flere setninger. Ellers er det fort gjort at en ny setning havner utenfor for-løkken. Noen liker å ha alt på én linje hvis det er kun en kort setning. Da droppes blokken.

Variabelen som initialiseres kalles generelt en «tellevariabel» (eng: counter). En står helt fritt til å velge navn på den, men hvis for-løkken arbeider i en tabell, brukes ofte navn som *i*, *j*, *k*, *m* og *n*. Det kommer nok av at de samme bokstavene brukes i summeformler i matematikk. Bokstaven *l* brukes vanligvis ikke siden den i mange tegnsett ligner for mye på sifferet 1.

Det kan være situasjoner der en trenger flere «tellevariabler». Da må det være komma mellom dem og ikke semikolon. I flg. eksempel kopieres innholdet av tabellen *a* over i tabellen *b*, men i motsatt rekkefølge:

```
int[] a = {1,2,3,4,5};
int[] b = new int[a.length];      // like lang som a

for (int i = 0, j = b.length - 1; i < a.length; i++, j--)
{
  b[j] = a[i];    // kopierer fra a til b
}

System.out.println(Arrays.toString(b));  // Utskrift: [5, 4, 3, 2, 1]
```

Programkode* E.1 *d)

Rekkevidden (eng: scope) til en «tellevariabel» er for-løkkens blokk. I noen situasjoner kan det være gunstig å bryte av (break) en for-løkke når noe spesielt inntreffer. Hvis vi da oppretter tellevariabelen foran for-løkken, vil vi kunne få tak i dens verdi etterpå:

```
int i = 0;
for ( ; i < a.length; i++)
{
  // programsetninger

  if <et eller annet> break;
}

// Nå kan vi få tak i verdien til i
```

Programkode* E.1 *e)

Dette betyr at en eller flere av for-løkkens deler kan være tomme, dvs. ingen kode. Det mest ekstreme er at hverken initialiseringen, betingelsen eller oppdateringen har kode:

```
for ( ; ; )
{
  // kropp
}
```

Denne konstruksjonen omtales som en «evig løkke». Det er situasjoner der det er gunstig å sette i gang en løkke på denne måten. Men da må det finnes kode blant programsetningene i kroppen som gjør at løkken garantert brytes på et eller annet tidspunkt.

For-løkker er så vanlige at de fleste utviklingsverktøy har maler (eng: templates) for dem. Hvis en bruker NetBeans, har en tabell med navn *tabell* og trenger en for-løkke som skal arbeide i den, kan en først skrive **fori** og så trykke tabulator-tasten. Da kommer dette:

```
for (int i = 0; i < tabell.length; i++)
{
  int j = tabell[i];

}
```

I Eclipse kan en skrive **for** og så trykke CTRL+mellomrom. Da kommer det tilbud om flere maler. Velges *for - iterate over array*, kommer dette:

```
for (int i = 0; i < tabell.length; i++)
{

}
```

Netbeans og Eclipse har flere maler enn dette for for-løkker.

E.2 For-alle-løkker

En for-alle-løkke (eng: for-each loop) brukes til å traversere en hel datastruktur. Det gjelder:

- tabeller
- datastrukturer som er `Iterable`

En generell for-alle-løkke ser slik ut:

```
for (<datatype> <navn> : <objekt>)
{
  // programsetninger
}
```

Dette kan vi lese slik: for alle verdier **navn** av typen **datatype** fra **objekt** osv. Her må objekt være en tabell av den gitte typen eller en instans av en generisk klasse som er `Iterable`.

Eksempel med en tabell:

```
int[] a = {1,2,3,4,5,6,7,8,9,10};

for (int k : a) System.out.print(k + " ");

// Utskrift: 1 2 3 4 5 6 7 8 9 10
```

Programkode* E.2 *a)

Eksempel med en liste:

```
List<Character> liste = new LinkedList<>();

liste.add('A'); liste.add('B'); liste.add('C');

for (Character c : liste) System.out.print(c + " ");

// Utskrift: A B C
```

Programkode* E.2 *b)

`LinkedList` er `Iterable` siden `LinkedList` implementerer `List`, den arver (eng: extends) `Collection` og `Collection` arver `Iterable`. Alle disse tre er grensesnitt (eng: interface). Når en klasse er `Iterable` har den garantert metoden *iterator*(), dvs. en metode som returnerer en `Iterator`. Det er den som brukes implisitt i for-alle løkken.

Java har mange `Iterable`-klasser. F.eks. flg. fra `java.util`: `ArrayDeque`, `ArrayList`, `HashSet`, `LinkedHashSet`, `LinkedList`, `PriorityQueue`, `Stack`, `TreeSet` og `Vector`.

Eksempel med tabell og mengde (eng: set):

```
int[] a = {5,9,2,4,7,10,8,1,3,6};              // en tabell

Set<Integer> s = new TreeSet<>();              // en mengde (TreeSet)

for (int k : a) s.add(k);                      // fra tabell til mengde

for (int k : s) System.out.print(k + " ");     // skriver ut

// Utskrift: 1 2 3 4 5 6 7 8 9 10
```

Programkode* E.2 *c)

E.3 While-løkker

En while-løkke har en *betingelse* og en *kropp* (eng: body) og ser generelt slik ut:

```
while (<betingelse>)
{
  // kropp
}
```

While-løkken går så lenge som betingelsen er oppfylt. Alt vi kan få til med en for-løkke kan vi få til med en while-løkke, og omvendt. *Programkode* E.1 *a)* viser en for-løkke som traverserer en tabell. Dette løses slik ved hjelp av en while-løkke:

```
int i = 0;

while (i < a.length)
{
  // kropp

  i++;
}
```

Programkode* E.3 *a)

Som nevnt i *Avsnitt* E.1, kan det av og til være gunstig å sette i gang en «evig løkke». Da må selvfølgelig løkkens «kropp» inneholde kode som gjør at løkken før eller senere brytes (break eller return). En «evig» while-løkke lages slik:

```
while (true)
{
  // kropp
}
```

Ved fil-lesing brukes ofte en while-løkke. I flg. eksempel finner vi antall linjer på en tekstfil:

```
FileReader fil = new FileReader("fil.txt");
try (BufferedReader inn = new BufferedReader(fil))
{
  String s;  // skal romme en linje fra filen
  int antall = 0;

  while ((s = inn.readLine()) != null)
  {
    antall++;
  }
```

```
  inn.close();

  // variabelen antall inneholder nå antall linjer på filen
}
```

Programkode* E.3 *b)

Konstruksjonen over er litt spesiell. Det er `linje != null` som er betingelsen, men det utføres en innlesing før den testes. Dette kunne vært løst slik med en for-løkke:

```
int antall = 0;

for (String s = inn.readLine(); s != null; s = inn.readLine())
{
  antall++;
}
```

E.4 Do-while-løkker

I både for-løkker og while-løkker sjekkes betingelsen før programsetningene i løkkens kropp utføres. I noen situasjoner kan det være aktuelt å utføre setningene én gang før betingelsen sjekkes. En do-while-løkke er laget for det formålet. En slik løkke består av en *kropp* (eng: body) og en *betingelse* og ser generelt slik ut:

```
do
{
  // kropp
}
while (<betingelse>);
```

I *Programkode* E.3 *b)* brukte vi en while-løkke til å telle opp antall linjer på en tekstfil. Dette kan også løses ved en do-while-løkke. Legg merke til at variabelen *antall* nå blir initiert til -1. Det kommer av at setningen *antall*++ blir utført én gang selv om filen ikke skulle ha noen linjer, dvs. den er tom.

```
FileReader fil = new FileReader("fil.txt");
try (BufferedReader inn = new BufferedReader(fil))
{
  String s;  // skal romme en linje fra filen
  int antall = -1;

  do
  {
    s = inn.readLine();
    antall++;
  }
  while (s != null);

  inn.close();

  // variabelen antall inneholder nå antall linjer på filen
}
```

Programkode* E.4 *a)