Java-ohjelmointi
Opas ammattimaiseen osaamiseen

Luku 4

Ehto- ja toistolauseet

Lisäys edelliseen lisenssiin: Kalvojen muokkaaminen on sallittu vain opettajille, joiden kursseilla käytetään kurssikirjana Tuloksellinen Java-ohjelmointi - tai Java ohjelmointi opas ammattimaiseen osaamiseen -kirjaa.

Ehto- ja toistolauseet

- Uutena asiana opetellaan ohjelmointilauseet / -rakenteet, jotka mahdollistavat:
  - Päätösten tekemisen ohjelman suorituksen aikana (esim. kyllä/ei)
  - Samojen lauseiden (ohjelmalohkon) toistamisen useammin kuin kerran
- Kirjassa nämä sijaitsevat luvussa 4:
  - 4.1 Johdanto
  - 4.2 Ehtolauseet
  - 4.3 Toistolauseet
  - 4.4. Ehto- ja toistolause -esimerkki

Missä ollaan?

ehtolausekkeet

if-ehtolause

Muut ehtolauseet

while-lause

Muut toistolauseet

Ehtolausekkeet

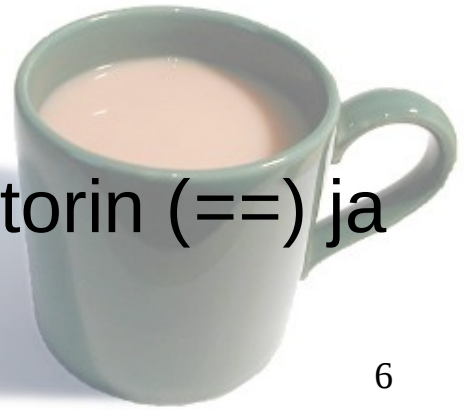
- Lauseiden suoritusjärjestys metodin sisällä on oletusarvoisesti lineaarinen (ylhäältä alas). Tällaista suoritusjärjestystä kutsutaan peräkkäisrakenteeksi.
- Ehto- ja toistolauseet mahdollistavat valinnan suoritetaanko ohjelmalohko vai ei ja kuinka monta kertaa.
- Valinnat perustuvat totuusarvoisiin (boolean) ehtolausekkeisiin (eli ehtoihin), jotka saavat aina arvon **`true`** tai **`false`**.
- Lauseiden suoritusjärjestystä kutsutaan kontrollivirraksi (flow of control).

Vertailuoperaattorit

- Ehtolausekkeet käyttävät Javan vertailuoperaattoreita, jotka kaikki palauttavat arvon tosi tai epätosi:

== yhtä suuri kuin
!= erisuuri kuin
< pienempi kuin
> suurempi kuin
<= pienempi- tai yhtä suuri kuin
>= suurempi- tai yhtä suuri kuin

- Huomaa ero yhtä suuruusoperaattorin (==) ja sijoitusoperaattorin välillä (=)!



Loogiset operaattorit

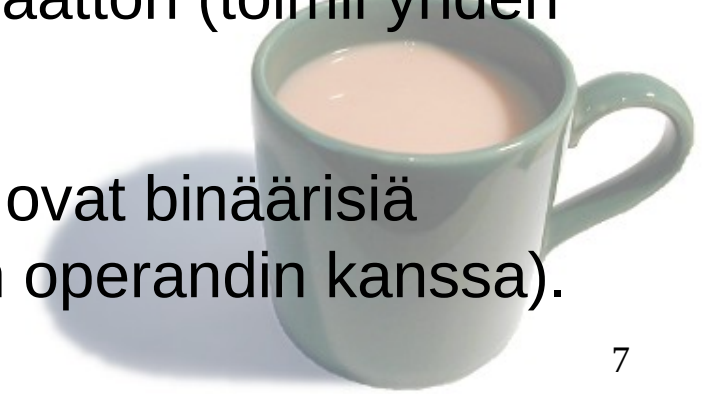
- Ehtolausekkeet voivat käyttää myös seuraavia loogisia operaattoreita:

  ! NOT (EI)

  && AND (JA)

  || OR (TAI)

- Operandeina on ehtolauseita.
- Looginen EI on unaarinen operaattori (toimii yhden operandin kanssa).
- Loogiset JA sekä looginen TAI ovat binäärisiä operaattoreita (toimivat kahden operandin kanssa).



Looginen NOT

- NOT-operaatiota kutsutaan myös negaatioksi tai komplementiksi.
- Jos operandi **a** on tosi, silloin **!a** (NOT a) on epätosi.
- Jos operandi **a** on epätosi, silloin **!a** on tosi.
- Tämä voidaan esittää totuusarvotaulukon avulla seuraavasti:

a	!a
true	**false**
false	**true**

Looginen NOT

- Loogista NOT-operaattoria voidaan käyttää esimerkiksi merkkijonon erisuuruusvertailuun.

```
String etunimi = ”Matti”;

if(!etunimi.equals(”Anna”)) {
    System.out.println(”Eri nimiset!”);
}
```

Looginen AND ja looginen OR

- Looginen AND (JA) lauseke

a && b

on tosi, jos sekä **a** että **b** ovat tosia, muutoin lauseke on epätosi.

- Looginen OR (TAI) ilmaisu

a || b

on tosi, jos **a** on tosi tai **b** on tosi tai molemmat ovat tosia, muutoin lauseke on epätosi (eli silloin kun **a** ja **b** ovat molemmat epätosia).

Loogiset operaattorit

- Loogisia operaattoreita käyttäen saadaan aikaiseksi turhankin monimutkaisia lausekkeita.

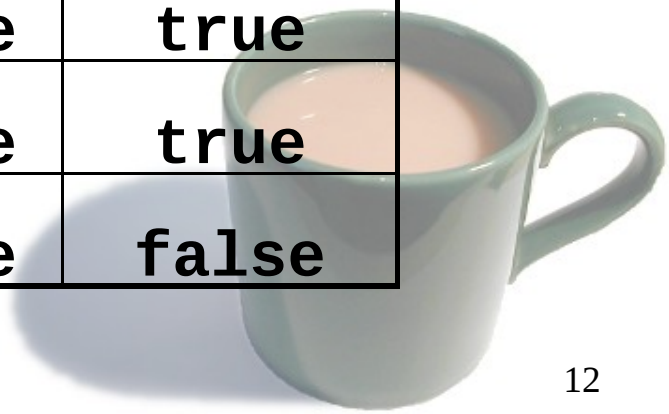
```
if (saldo > otto + 500 && !tilivirhe) {
   System.out.println (”Rahat riittävät.”);
}
```

- Sulkujen käyttö suoritusjärjestyksen selventämiseksi on aina suositeltavaa!

Loogiset operaattorit

- Totuusarvotaulukosta nähdään kaikki mahdolliset tilanteet esimerkiksi kahden operandin tilanteessa
- Koska **&&**- ja **||**-operaattoreilla on kaksi operandia, on operandien yhdistelmiä olemassa neljä.

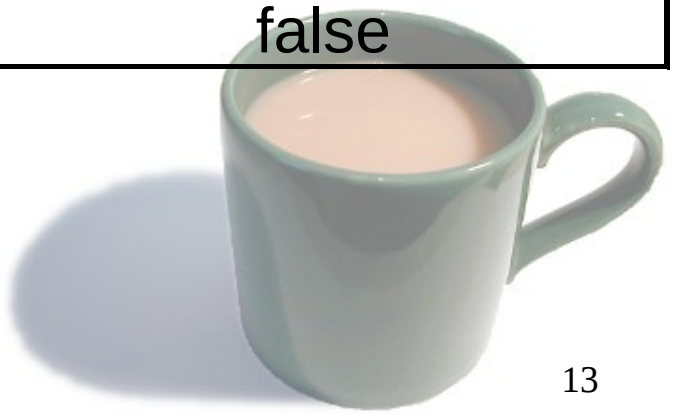
a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false



Totuusarvotaulukko

- Minkä tahansa lausekkeen voi evaluoida totuusarvotaulukon avulla.

`saldo > otto + 500`	`tilivirhe`	`!tilivirhe`	`saldo > otto + 500 && !tilivirhe`
true	true	false	false
true	false	true	true
false	true	false	false
false	false	true	false



Missä ollaan?

ehtolausekkeet

if-ehtolause

Muut ehtolauseet

while-lause

Muut toistolauseet

Ehtolauseet

- Ehtolause mahdollistaa valinnan suoritetaanko ehtoon liittyvä ohjelmalohko.
- Ehtolauseita kutsutaan myös valintalauseiksi (selection statements).
- Javan ehtolauseet ovat:
  - **`if`**-lause
  - **`if-else`** –lause
  - **`if - else if`** -lause
  - **`switch`**-lause.

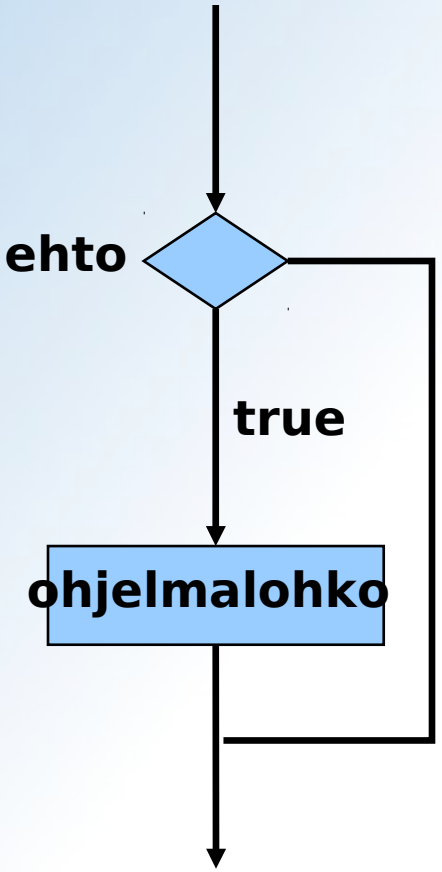
`if`-lause

- **`if`**-lauseella on seuraava syntaksi:

```
if(ehto) {
   lause1;
   lause2;
}
```

- **`if`** on varattu sana.
- ***ehto*** on totuusarvoksi evaluoituva lauseke.
- Jos ***ehto*** on tosi, suoritetaan **`if`**-lauseeseen liittyvä ohjelmalohko.
- Jos ***ehto*** on epätosi, ohjelmalohko jätetään suorittamatta.

`if`-lauseen logiikka



`if`-lause

- Esimerkki `if`-lauseesta:

```
if (saldo < 0) {
    System.out.println (”Tilillä ei ole katetta!”);
}
System.out.println(”Valmis.”);
```

- Ensin tutkitaan ehto :`saldo`-muuttuja on joko pienempi kuin nolla tai ei.
- Jos ehto on tosi (saldo on alle nollan), `if`-lauseeseen liittyvä ohjelmalohko suoritetaan – jos näin ei ole, lohko ohitetaan.
- Oli tilanne kumpi tahansa, seuraavaksi tulostetaan merkkijono ”Valmis”.
- Katso myös kirjan esimerkki 4.1.

`if`-lause

- Mitä seuraavat lauseet tekevät?

```
if (ylaraja < 0) {
   ylaraja = 0;
}
```

Sijoittaa muuttujaan `ylaraja` arvon nolla, jos muuttujan tämänhetkinen arvo on pienempi kuin nolla.

```
if (saldo != otot + panot) {
   tilivirhe = true;
}
```

Sijoittaa muuttujaan tilivirhe arvon `true`, jos `saldo`-muuttujan arvo ei ole sama kuin `otot`- ja `panot`-muuttujien summa.

- Aritmeettiset operaatiot suoritetaan ennen yhtäsuuruus- ja vertailuoperaatioita. Sulkujen käyttö asian selventämiseksi on kuitenkin aina suositeltavaa!

Laiska evaluointi

- AND- ja OR-operaattorien prosessointi on “laiskaa”.
- Jos ensimmäinen operandi riittää ratkaisemaan lopputuloksen, seuraavaa operandia ei huomioida ollenkaan.

```
if (nimi != null && nimi.equals(”Bill”)) {
   System.out.println (”Terve Bill!");
}
```

- Tästä on hyötyä esim. edellisessä tilanteessa: ohjelma ei kaadu vaikka merkkijonon nimi arvo olisi **`null`**.

Missä ollaan?

ehtolausekkeet

if-ehtolause

Muut ehtolauseet

while-lause

Muut toistolauseet

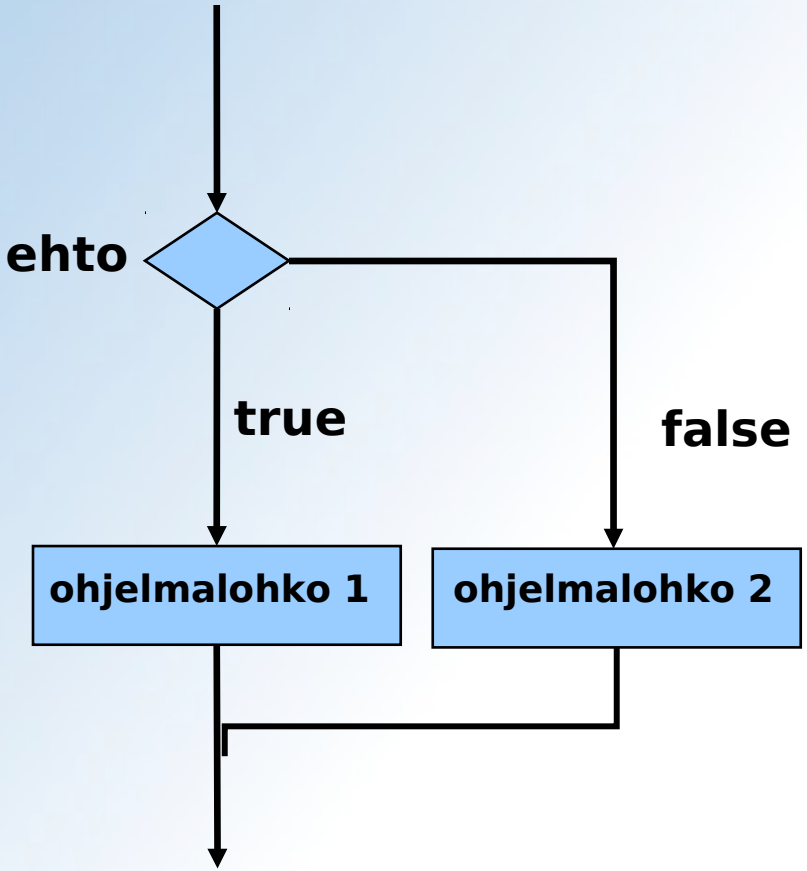
`if-else` -lause

- **`else`**-lause lisätään **`if`**-lauseeseen, kun halutaan epätosi-vaihtoehdolle erillinen käsittely:

```
if(ehto) {
  lause1;
} else {
  lause2;
}
```

- Jos ***ehto*** on tosi, ***lause1*** (lohko 1) suoritetaan.
- Jos ***ehto*** on epätosi, ***lause2*** (lohko2) suoritetaan.
- Katso myös kirjan esimerkki 4.2.

`if-else` -lauseen logiikka



`if-else` -esimerkki

- **`if-else`** –lausetta tarvitaan esimerkiksi set-metodien validiteetti tarkistuksissa.
- Miksi validiteettitarkistuksia tarvitaan?

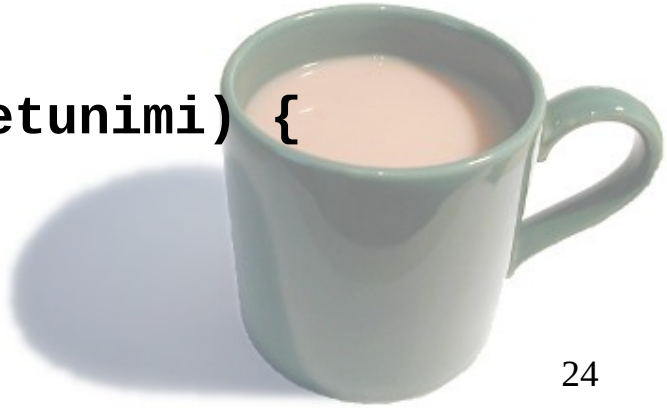
Esimerkki:

Luokassa Oppilas (kirja s.43) on attribuutti

```
private String etunimi;
```

Toteutetaan aluksi yksinkertaisin mahdollinen set-metodi (jonka myös Eclipse osaa generoida):

```
public void setEtunimi(String etunimi) {
   this.etunimi = etunimi;
}
```



`if-else` -esimerkki

Metodi täyttää tärkeimmän tehtävänsä, eli luokan ulkopuolelta saadaan tallennettua olion attribuuttiin haluttua tietoa esimerkiksi seuraavasti:

```
Oppilas oppilas1 = new Oppilas();

oppilas1.setEtunimi("Ville");
```

Kysymys:

Mitä tapahtuu, mikäli set-metodia kutsutaan esimerkiksi seuraavilla arvoilla?

```
oppilas1.setEtunimi("");

oppilas1.setEtunimi(null);
```

`if-else` -esimerkki

- Edellä toteutettu pelkästään suoran sijoituksen sisältävä set-metodi alkaa lähestyä seuraavaa tilannetta:
  - Mikäli olion kapselointi rikottaisiin määrittelemällä attribuutti suojatun **`private`**-tyypin sijaan suoraan **`public`**-tyyppisenä eli julkisena (älä koskaan tee näin!), olisi mahdollista tehdä luokan ulkopuolelta suoria sijoituksia:

```
oppilas1.etunimi=null;
```

vrt.

```
oppilas1.setEtunimi(null);
```

`if-else` -esimerkki

- Ero em. kapseloinnin rikkomiseen on lähinnä siinä, että suoran sijoituksen sijaan tulee tietoisesti kutsua set-metodia.
- Suora sijoitus on helpompi kirjoittaa "vahingossa", **`public`**-tyyppiseksi muunto lisäisi virhealttiutta entisestään.
- Haluamme kuitenkin ohjelmamme toimivan aina oikein, joten set-metodin tulee hylätä virheelliset arvot!
- Ratkaisu tähän ovat validiteettitarkistukset.

`if-else` -esimerkki

- Edellä olevassa tilanteessa etunimeksi eivät kelpaa ainakaan **`null`** eikä tyhjä merkkijono.
- Saadaan seuraavanlainen validiteettitarkistukset sisältävä set-metodi.

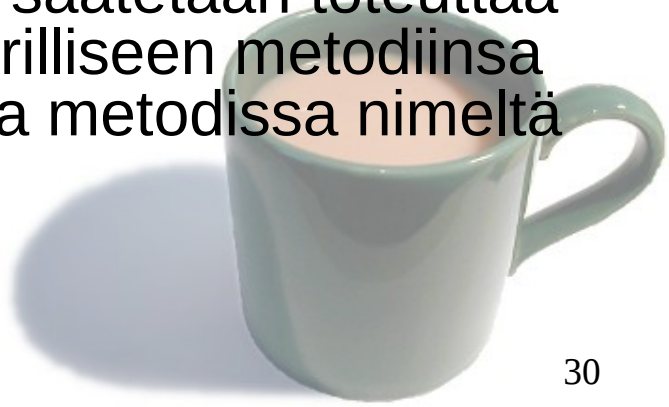
```
public void setEtunimi(String etunimi) {
    if(etunimi != null && etunimi.length() > 0) {
        this.etunimi = etunimi;
    } else {
        System.out.println("Etunimi ei saa olla
                tyhjä!");
    }
}
```

`if-else` -esimerkki

- Tämän jälkeen set-metodia on hyödynnettävä muualla koodissa, jotta duplikaattikoodilta vältytään.
- Esimerkiksi parametrillisessa kontruktorissa tulee suoran sijoitukset **`this.etunimi = etunimi`** sijaan kutsua set-metodia: **`setEtunimi(etunimi)`**.

`if-else` -esimerkki

- Edellä esitelty tapa toteuttaa validiteettitarkistukset riittää meille toistaiseksi.
- Todellisuudessa esiintyy kuitenkin seuraavia eroja edelliseen:
  - Virheilmoituksia ei tulosteta suoraan konsolille vaan lokitiedostoon esimerkiksi paketin **`java.util.logging`** avulla (katso kirjan luokka **`TiedostoTyokalut`** s. 74).
  - JavaBeaneja käyttävissä sovelluskehyksissä (esim. Struts) validiteettitarkistukset saatetaan toteuttaa set-metodien sijaan omaan erilliseen metodiinsa (esim. **`ActionForm`**-luokassa metodissa nimeltä **`validate`**).



`If - else-if` -lause

- **`If – else-if`** on rakenne, jossa voidaan huomioida useita ehtoja.

```
if(ehto1) {
  lause1;
} else if(ehto2) {
  lause2;
} else if(ehto3) {
  lause3;
} else {
  lause4;
}
```

`If - else-if` -lause

- **`else-if`** -osia voidaan kirjoittaa ensimmäisen **`if`**-osan jälkeen haluttu määrä
- Jokaisella **`else-if`** –osalla on oma ehtolausekkeensa.
- Loppuun voidaan kirjoittaa **`else`**-osa.

If - else-if -lause

- Esimerkki: Tutkitaan mikä maa on kyseessä ja asetetaan maan mukainen valuutta.

```
if(maa.equals(”Suomi”)) {
valuutta = ”Euro”;
} else if(maa.equals(”Ruotsi”)) {
valuutta = ”Kruunu”;
} else if(maa.equals(”USA”)) {
valuutta = ”Dollari”;
} else {
   valuutta = null;
}
```

- Mitä jos muuttujan maa arvo onkin null?
- Kirjassa lisäksi esimerkki 4.3.

`switch`-lause

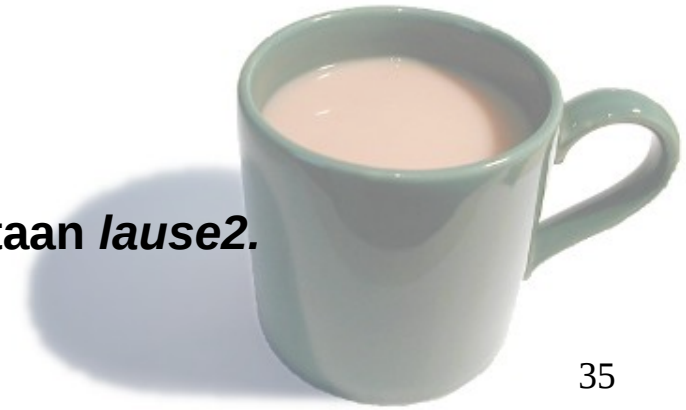
- **`switch`**-lauseella saadaan toteutettua monivalintarakenne (mahdollista myös **`if – else-if`** –rakenteena)
- **`switch`**-lauseelle luetellaan eri arvovaihtoehdot, joista suoritetaan se joka täsmää valintaehtoon.
- **`switch`**-lauseen ehdoksi kelpaa vain primitiivityypiksi evaluoituva lauseke.

`switch`-lause

- **`switch`-lauseen syntaksi on seuraava:**

```
switch ( ehto ) {
 case arvo1 :
    lause1;
 case arvo2 :
    lause2;
 case arvo3 :
    lause3;
 default:
lause4;
}
```

- **`switch` ja `case` ovat varattuja sanoja.**
- **Jos *ehto* täsmää arvoon *arvo2*, suoritetaan *lause2*.**



`switch`-lause

- **`break`**-lause tarvitsee sijoittaa lähes aina **`case`**:n viimeiseksi lauseeksi.
- **`break`**-lauseesta seuraa kontrollin siirtyminen **`switch`**-lauserakenteen loppuun.
- Jos **`break`**-lausetta ei käytetä, kontrolli ”valuu” järjestyksessä seuraavaan **`case`**en.
- **`default`**:lla voidaan määritellä lauseet jotka suoritetaan mikäli yksikään **`case`** ei toteutunut.

switch-lause

- Esimerkki **switch**-lauseesta:

```
switch (valinta) {
   case ’T’:
      tallenna();
      break;
   case ’U’:
      luoUusi();
      break;
   case ’L’:
      lopeta();
      break;
}
```

- Katso myös kirjan esimerkki 4.4.

Missä ollaan?

ehtolausekkeet

if-ehtolause

Muut ehtolauseet

while-lause

Muut toistolauseet

Toistolauseet

- Toistolauseet mahdollistavat tietyn ohjelmalohkon suorittamisen useita kertoja.
- Toistolauseesta käytetään usein nimitystä silmukka tai luuppi.
- Myös toistolauseita kontrolloidaan ehtolausekkeiden avulla.
- Javassa on kolmenlaisia toistolauseita:
  - **`while`**-lause
  - **`do-while`** -lause
  - **`for`**-lause

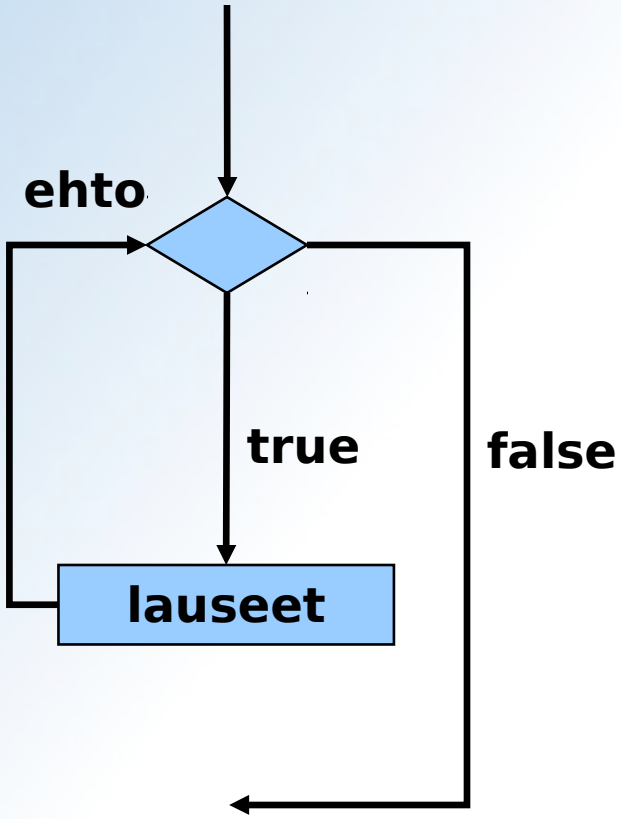
`while`-lause

- **`while`**-lauseella on seuraava syntaksi:

```
while ( ehto ) {
  lauseet;
}
```

- Jos ***ehto*** on tosi (**`true`**), lauseet (ohjelmalohko) suoritetaan
- Lauseiden suorittamisen jälkeen ehto evaluoidaan uudestaan.
- Jos ehto on edelleen tosi, lauseet suoritetaan taas.
- Lauseita suoritetaan toistuvasti niin kauan kunnes ehdosta tulee epätosi (**`false`**).

while-lauseen logiikka



while-lause

- Esimerkki **while**-lauseesta:

```
Scanner scan = new Scanner(System.in);
int luku = scan.nextInt();
while (luku > 0) {
   System.out.println(luku);
   luku = scan.nextInt();
}
```

- Jos **while**-lauseen ehto on heti alussa epätosi, ei lauseeseen liittyvää ohjelmalohkoa suoriteta kertaakaan.
- Katso myös kirjan esimerkki 4.5.

Ikuiset silmukat

- **`while`**-silmukan ohjelmalohkossa on pakko jossain vaiheessa tehdä ehdosta **`false`**-arvoinen.
- Jos näin ei käy, suoritetaan silmukkaa ikuisesti.
- Tämä on yleinen looginen virhe ohjelman suunnittelussa / toteutuksessa.

Ikuiset silmukat

- Esimerkki ikuisesta silmukasta eli “ikiluupista”:

```
int laskuri = 1;
while (laskuri < 100) {
   System.out.println(laskuri);
   laskuri = laskuri - 1;
}
```

Sisäkkäiset silmukat

- Silmukoita voidaan kirjoittaa useita sisäkkäin, eli silmukan ohjelmalohko voi sisältää toisen silmukan.

```
int ulompi = 10;
while (ulompi > 0) {
    int sisempi = 10;
    while (sisempi > 0) {
        System.out.println(”Sisempi”);
        sisempi = sisempi - 1;
    }
    System.out.println(”Ulompi”);
    ulompi = ulompi - 1;
}
```

Missä ollaan?

ehtolausekkeet

if-ehtolause

Muut ehtolauseet

while-lause

Muut toistolauseet

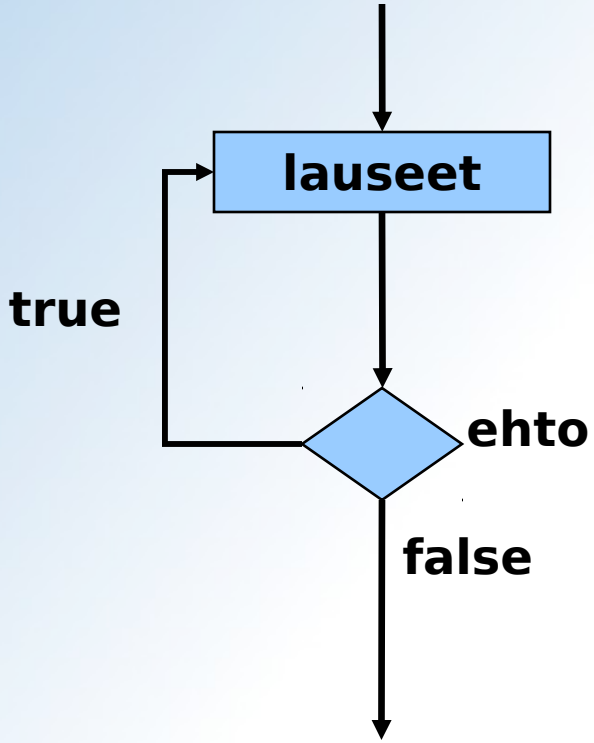
`do-while` -lause

- **`do-while`** -lauseella on seuraava syntaksi:

```
do {
  lauseet;
} while(ehto);
```

- Lauseet (ohjelmalohko) suoritetaan ensin kertaalleen, sitten ehto evaluoidaan.
- Lauseet suoritetaan toistuvasti kunnes ehdosta tulee epätosi (**`false`**).

`do-while` -lauseen logiikka



do-while -lause

- Esimerkki **do-while** -lauseesta:

```
boolean totuus = false;
do {
    System.out.println(”suoritus”);
} while (totuus == true);
```

- Kuinka monta kertaa yo esimerkin tulostuslause suoritetaan?
- Katso myös kirjan esimerkki 4.6.

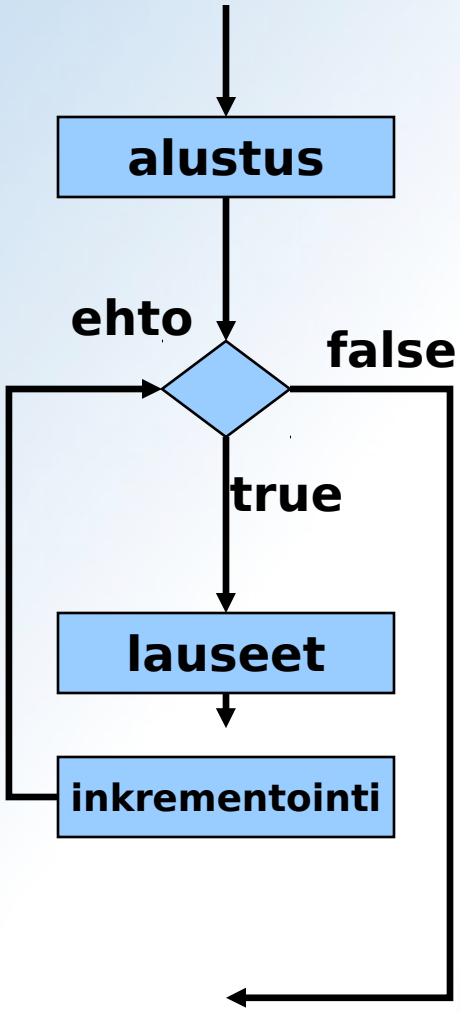
`for`-lause

- `for`-lauseella on seuraava syntaksi:

```
for(alustus; ehto; inkrementointi) {
        lauseet;
 }
```

- ***Alustus*** suoritetaan kerran ennen kuin silmukka alkaa.
- ***Inkrementointi***-osa suoritetaan jokaisen toiston lopussa.
- ***Lauseet*** suoritetaan kunnes ***ehto*** saa arvo `false`.

`for`-lauseen logiikka



`for`-lause

- **`for`**-lause on toiminnallisesti identtinen seuraavan **`while`**-lauseen kanssa:

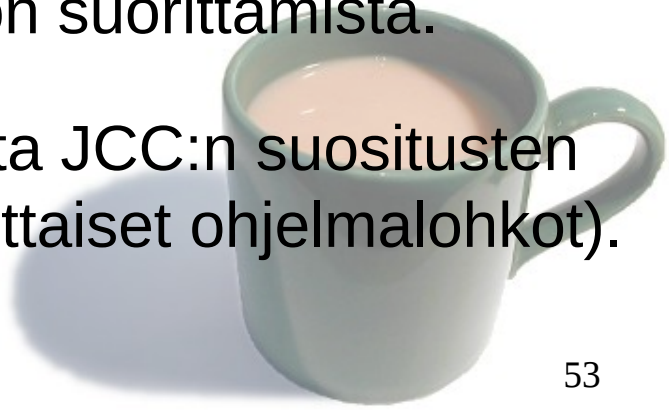
```
alustus;
while(ehto) {
  lauseet;
  inkrementointi;
}
```

`for`-lause

- Esimerkki **`for`**-lauseesta:

```
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

- Alustusosaa voidaan käyttää muuttujan esittelemiseen.
- Kuten **`while`**-silmukassa, **`for`**-silmukassa ehto testataan ennen silmukan rungon suorittamista.
- Käytä mieluiten aina aaltosulkeita JCC:n suositusten mukaisesti (myös yhden rivin mittaiset ohjelmalohkot).



`for`-lause

- **`for`**-lause sopii hyvin tilanteisiin, jossa suoritus halutaan toistaa tietty määrä kertoja (toistokerrat voidaan laskea tai määrittää etukäteen).
- **`for`**-lause on ilmaisultaan tiivis.
- Kaikki **`for`**-lauseet voidaan kirjoittaa myös **`while`**-lauseina.

Ehto- ja toistolause-esimerkki

```
import java.util.Scanner;
/**
 * Coca-cola automaatti.
 * @author Jukka Juslin, Jukka Harju
 */
public class CokisAutomaatti {
    private int pullojenMaara;
    private double pullonHinta;
    public CokisAutomaatti() {
        pullojenMaara = 30;
        pullonHinta = 2.0;
    }
    public int annaKolaa(double rahaaAnnettu) {
        System.out.println("Vaihtorahaa saat takaisin: "
            + (rahaaAnnettu % pullonHinta));
        return (int) (rahaaAnnettu / pullonHinta);
    }
```

Ehto- ja toistolause-esimerkki

```
    public void kaynnistaAutomaatti() {
        double rahaaAnnettu = 0;
        int saitPulloja = 0;
        Scanner scan = new Scanner(System.in);
        while (pullojenMaara > 0) {
            System.out.println("Anna rahaa: ");
            rahaaAnnettu = scan.nextDouble();
            if (rahaaAnnettu >= pullonHinta) {
                saitPulloja = annaKolaa(rahaaAnnettu);
                pullojenMaara = pullojenMaara - saitPulloja;
                if (pullojenMaara > 0) {
                    System.out.println("Sait "
                        + saitPulloja + " pulloa.");
                } else {
                    System.out.print(
                        "Sold out. Pullot loppu, ");
                    System.out.println("soita 09 11.");
                }
            } else {
                System.out.println("Raha ei riittänyt.");
            }
        }
    }
    public static void main(String[] args) {
        CokisAutomaatti tomaatti = new CokisAutomaatti();
        tomaatti.kaynnistaAutomaatti();
    }
}
```