

TYÖ: RUOKAPÄIVÄKIRJA

Sisällys

TYÖ: RUOKAPÄIVÄKIRJA.....	1
Sovelluksen tarkempi idea.....	2
Työn rajaukset.....	2
Laskentakaavat	2
Sovelluksen tärkeimmät tietokentät	3
Sovellukset relaatiot	4
Sovellukseen tehty luokat.....	4
AlustaTietokanta.java	4
Enegia.java.....	4
Henkilo.java	5
Henkilotiedot.java	5
Levytoiminta.java.....	5
Paivalaskuri.java	6
Paivakirja.java	6
Ruokapaivakirja.java	7
Tapahtuma.java.....	9
Tuote.java.....	9
Tuotetiedot.java	9
Yhteenveto.java.....	10



Lähtökohtana oli toteuttaa vapaasti jatkojalostettavissa oleva idea ruokapäiväkirjasta, joka voitaisiin siirtää esimerkiksi kännyköihin. Kännyköiden kameraa hyväksikäyttäen voidaan esim. GTIN/EAN-koodit (viivakoodi) ja muut kameran avulla luettavat koodit tulkita numerotiedoksi, joka yksilöi tuotteen.

Sovelluksen tarkempi idea

Tavoitteena oli toteuttaa ruokapäiväkirja, jossa käyttäjä voi lukea 13-merkkisestä GTIN-koodista (joko tuotteesta itsestään tai erikseen tulostetusta tarrasta) tuotteen yksilöivän tiedon numeroina (tai viivakoodina näppäimistöön liitetystä viivakoodilukijasta), jonka perusteella sovellus osaa hakea tietokannasta kyseiseen tietoon perustuen ravintosisällöt tarkkuudella hiilihydraatit, proteiinit, rasvat ja energia.

Päiväkirjaa käytetään reaaliaikaisesti sen mukaisesti, miten syödään (esimerkiksi jos päiväkirja toimisi kännykässä). Päiväkirjaa voidaan myös täydentää jälkikäteen valitsemalla tuotteet ja syödyt määrät valikosta.

Päiväkirjasta muodostuu energian sisäänoton tiedot, joita verrataan WHO:n energiatarpeen ennusteyhtälöön. Toteutuneen energian sisäänoton ja ennusteyhtälön tiedoilla ennustetaan laihtuminen tai lihominen siten, että jokainen kilo rasvakudosta sisältää 7000 kcal. Esimerkiksi jos syö viikossa 3500 kcal vähemmän kuin energiatarve, laihtuminen tulisi olla 0,5 kg. Mitatun painon ja ennusteen mukaan voidaan toteuttaa laskelma, josta näkee minkä verran ennuste poikkeaa todellisuudesta.

Työn rajaukset

- Sovellus tuotettiin ilman varsinaista relaatiotietokantaa kuten MySQL tms.
- Tietokantana toimii kolme tiedostoa, jossa tarvittavat tietokentät ja esimerkkidatat.
- Tiedostossa on testaamista varten on kerätty n. 70 yleistä tuotetta ja niiden ravintoarvot (löytyy esimerkiksi K-marketin sivuilta).

Laskentakaavat

Sovellukselle syötetään käyttäjän sukupuoli, nimi, ikä, pituus ja paino, joiden perusteella henkilöprofiiliin lasketaan painon mukaisesti ¹WHO:n ennusteyhtälön mukaiset energia-arvot käyttämällä kaavaa:

Perusaineenvaihdunta BMR (ns. koomakulutus, eli tämän keho kuluttaa vaikka et tekisi mitään):

BMR laskentakaava miehille:

$$\text{BMR} = 66 + (13.7 \times \text{paino kilo}) + (5 \times \text{pituus cm}) - (6.8 \times \text{ikä vuosina}).$$

BMR laskentakaava naisille:

$$\text{BMR} = 655 + (9.6 \times \text{paino kilo}) + (1.8 \times \text{pituus cm}) - (4.7 \times \text{ikä vuosina}).$$

Kokonaiskulutus²:

Kokonaiskulutus miehille:

$$\text{Päivittäinen KCAL} = \text{aktiivisuuskertoimen} \times (66 + (13.7 \times \text{paino kg}) + (5 \times \text{pituus cm}) - (6.8 \times \text{ikä vuosia})).$$

Kokonaiskulutus naisille:

$$\text{Päivittäinen KCAL} = \text{aktiivisuuskertoimen} \times (655 + (9.6 \times \text{paino kg}) + (1.8 \times \text{pituus cm}) - (4.7 \times \text{ikä vuosia})).$$

Aktiivisuuskertoimen 1-1,9 liikuntatottumusten mukaisesti.

Aktiivisuuskertoimet kokonaiskulutukseen:

- Ei mitään (makaa sängyssä koko päivän) = BMR x 1,0
- Tosi vähän BMR x 1,2
- Kevyt (1-3 kertaa viikossa) BMR x 1,375
- Keskimääräinen (3-5 kertaa viikossa) BMR x 1,55
- Raskas (6 kertaa viikossa) BMR x 1,725
- Duracell pupu BMR x 1,9

¹ Tunnetaan nimellä Calories Burned Calculator

² Olen käyttänyt kertoimia weightloss.about.com sivuston kaavoista (samat muualla)



Laihtumisen/lihomisen ennustaminen:

- Jokainen rasvakudoskilo sisältää 7000 kcal.
- Energiavaje laihtuttaa, energialisä lihottaa.

Sovelluksen tärkeimmät tietokentät

GTIN-koodista muodostuva numerosarja toimii tuotteen yksilöivänä tietona ja tietokannan yksilöllisenä tunnisteena. Sama tunniste tallentuu myös päiväkirjaan, joka toimii näin ollen relaationa kahden taulun välillä. GTIN-tunnisteet ovat globaalisti yksilöllisiä. GTIN-tunnisteesta lisätietoja [tässä linkissä](#). Mikäli tuotetta ei löydy ennestään, kysytään tuotteen tiedot ja lisätään tiedostoon.

Tuote-tila (erillinen tiedosto):

1. GTIN-koodi (13 numeroa =primääriavain)
2. Tuotteen nimi
3. Tuotteen energia per yksikkö
4. Yksikkö (per 100 g, per kpl, per annos)
5. Tuotteen sisältämä proteiinimäärä
6. Tuotteen sisältämä hiilihydraattimäärä
7. Tuotteen sisältämä rasvamäärä
8. Huomautuskenttä
9. Onko tuote aktiivinen.

Edellisistä muodostuu Tuote-olioita, joiden tunnisteena ja relaatioavaimena on GTIN-koodi.

Henkilötieto on käytännössä henkilön profiili, johon kohdistetaan Tuotetiedoissa mainittuja tuotteita ja myöhemmin Ruokapäiväkirjassa mainituista tapahtumista.

Henkilötieto-tila (erillinen tiedosto):

1. Koko nimi (30 kirjainta)
2. Sukupuoli (true=mies, false=nainen)
3. Ikä vuosina
4. Paino kg
5. Pituus cm
6. Aktiivisuuskerroin
7. Huomautus
8. Onko henkilö aktiivinen
9. Henkilönumero (relaatio)

Edellisistä muodostuu Henkilo-olioita, joiden relaatioavaimena on henkilönumero.

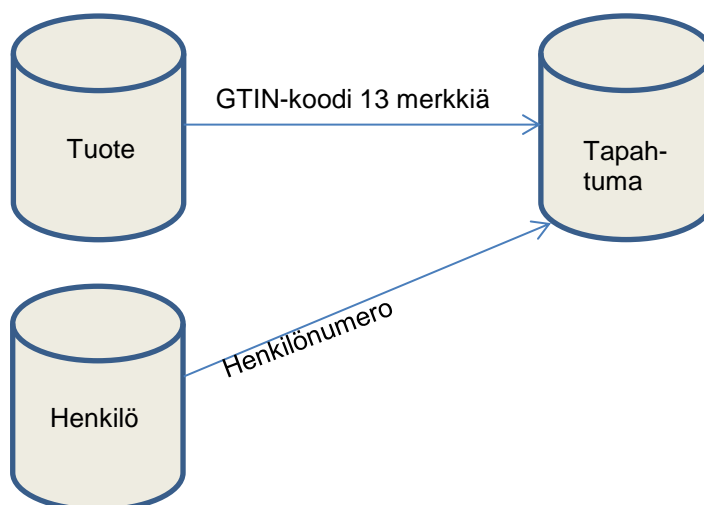
Ruokapäiväkirja kerää erilliseen tiedostoon mitä milloinkin on syöty (viittaa Tuote-tilaan relaationa GTIN-tunniste), minkä verran ja käyttäjän valitsemin ajankohdin punnituksen.

Tapahtuma-tila (erillinen tiedosto):

1. Päivämäärä tekstimuodossa pp.kk.vvv
2. Kello tekstimuodossa tt:mm:ss
3. Kelloaika Date-muodossa
4. Tapahtumanumero on järjestelmän millisekunnit (ei voi olla kahta samaa, tulevaisuuden varaus)
5. GTIN (EAN) numero, joka toimii relaatioavaimena
6. Henkilönumero, joka toimii relaatioavaimena
7. Syöty määrä
8. Mitattu uusi paino
9. Huomautuskenttä

Edellisistä muodostuu Tapahtuma-olioita.

Sovellukset relaatiot



Sovellukseen tehdyt luokat

Sovelluksessa on 12 luokkatiedostoa. Jokaisessa luokassa on kommentoitu koodia tarvittavilta osin. Luokista syntyy myös javadoc, mikäli haluaa tarkastella toimintaa sen avulla. Alla on kerrottu luokkien pääpiirteittäinen toiminta.

AlustaTietokanta.java

AlustaTietokannan tarkoitus on luoda tyhjä tuotteet, tapahtumat ja henkilöt- tiedostot (tietokannat³) ja alustaa niihin esimerkkihenkilöt (sovelluksen käyttöönottovaiheessa). Luokkaa kutsutaan Ruokapäiväkirja.java-luokan main:sta vain silloin, jos tiedostoja ei ole ennestään. Tiedostonimet ovat vapaasti parametrisoitavissa Ruokapäiväkirja.java-luokassa ja niistä luodaan aina .bak⁴-loppuinen vamuuskopio, kun sovellus tallentaa tiedostoja.

1. `public static void alustaTuoteTietokanta(String tietokanta)`
2. `public static void alustaTapahtumaTietokanta(String tietokanta)`
3. `public static void alustaHenkiloTietokanta(String tietokanta)`

Enegia.java

Enegia palauttaa WHO:n laskentakaavan mukaisen "koomakulutuksen" eli BMR:n ja aktiivisuuskertoimen avulla myös ennusteen kokonaiskulutuksesta. Tätä tietoa käytetään Yhteenveto-luokassa, kun lasketaan arvioita laihtumisesta.

Metodit:

1. `public static Double ennusteBMR (Double paino, Double pituus, int ika, boolean sukupuoli)`
2. `public static Double ennusteKokonaisKulutus (Double paino, Double pituus, int ika, boolean sukupuoli, Double aktiivisuus)`

³ Tietokanta on harhaanjohtava termi, koska kyseessä ei ole varsinainen tietokanta, josta olisi helppoa noutaa tietoja esimerkiksi päivämäärän perusteella esimerkiksi sql-kielellä.

⁴ Vapaasti määriteltävissä main-luokassa polkuineen. Oletuksena Ruokapäiväkirja juuressa.

Henkilo.java

Sisältää konstruktorit henkilötietojen luomiseen. Huomaa, että tyhjä⁵ konstruktori tarvitaan tiedostojen lukemiseksi levyltä muistiin.

Konstruktorit:

1. `public Henkilo(String inHenkiloNimi, Boolean inHenkiloSukupuoli, Integer inHenkilolka, Double inHenkiloPaino, Double inHenkiloPituus, Double inHenkiloAktiivisuus, String inHenkiloHuomautus, Boolean inHenkiloAktiivinen, Integer inHenkiloNumero)`
2. `public Henkilo()`

Henkilotiedot.java

Henkilotiedot sisältää java formin (kehys⁶), jolla voidaan luoda uusi henkilö tai luoda olemassa olevia henkilöitä. Tarvittavat rakenteet tietojen oikeellisuuden tarkistamiseen on jokaisen kentän action listenereissä, eli väärää tietoa ei voida tallentaa. Eri painikkeet aktivoituvat sen mukaan, mikä kulloinkin on sallittu toimenpide.

Sisältää vain action listenereihin⁷ tehtyjä logiikoita, joita ei tässä ole syytä luetella pituuden vuoksi.

Kuva 1 Henkilötietojen lisäys/muokkaus

Levytoiminta.java

Tällä luokalla luetaan levyltä tiedot taulukoihin ja tallennetaan tiedot taulukoista levyille. Huomaa, että jokainen konstruktori (olioineen) tulee olla `ns. serializable`, joka näkyy luokan esittelyssä implements `Serializable`-lauseesta.

Levytoiminta käsittelee virheitä try-catch-menetelmällä, eli mikäli jotain menisi pieleen, tulisi siitä ilmoitus (tätä käytetään monessa muussakin yhteydessä).

Käytetyt metodit:

1. `public static void tallenna(LinkedList taulukko, String tiedostoNimi)`
2. `public static LinkedList<Tuote> lueTuotteet(String tiedostoNimi)`
3. `public static LinkedList<Tapahtuma> lueTapahtumat(String tiedostoNimi)`
4. `public static LinkedList<Henkilo> lueHenkilot(String tiedostoNimi)`

⁵ Tämä tuli esille, kun testasin tiedostojen luentaa ja kirjoittamista. Asiaan voi vaikuttaa se, että jos sattuisi testausvaiheessa olemaan null-arvoinen tallennus, jolloin normaali konstruktori ei toimi.

⁶ Kehyksiä tehdään uusia ja tuhotaan sitä mukaa, miten nappuloita painetaan. Tämä ei ole järkevä ohjelmointitapa, mutta tässä vaiheessa en muuta osannut.

⁷ Action Listenerit ovat mm. eri painikkeisiin tehtyjä metodeja, joiden avulla muutetaan sovelluksen toimintaa. Esimerkkinä painamalla Muokkaa tuotetaan Muokkaa-painikkeen action listener-metodi.

Paivalaskuri.java

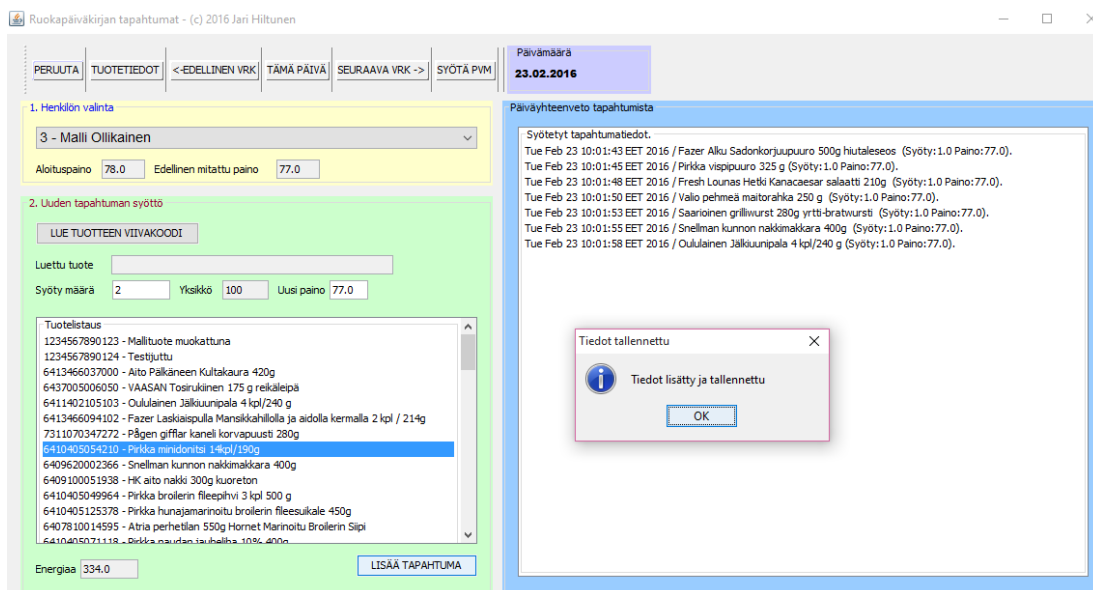
Paivalaskurilla lasketaan päivien välistä aikaa, lisätään ja vähennetään päiviä, muutetaan aikaa Date-muodosta String muotoon ja toisinpäin. Tällä pyrin välttämään virheitä koodissa, jossa syötetään⁸ päivämääriä, sillä päivien muuttaminen eri muotoon sisältää monenlaisia virhemahdollisuuksia. Toisaalta tämä menetelmä kuluttaa tietokannasta tilaa, eli pelkkä Date⁹-muotoinen tallennuskin olisi ollut mahdollisesti riittävä.

Metodit:

1. public static String lisaaPaivia(Date paivamaara, int paivia)
2. public static String poistaPaivia(Date paivamaara, int paivia)
3. public static String tamaPaiva()
4. public static String tamaKelloaika()
5. public static Date paivaDateksi(String paivamaara)
6. public static String datePaivaksi(Date paivamaara)
7. public static long stringPaivaero(String alkupvm, String loppupvm)

Paivakirja.java

Paivakirja on sovelluksen ydin. Sillä voidaan syöttää tiedot syödyistä tuotteista. Tuote voidaan valita joko listasta tai painamalla Lue tuotteen viivakoodi lukea käyttämällä viivakoodin¹⁰ lukijaa (joka palauttaa 13-numeroisen viivakoodin ja enterin). Mikäli tuotetta ei löydy ennestään, kysyy sovellus, lisätäänkö tuote tietokantaan ja siirtyy tietokannan ylläpitoon.



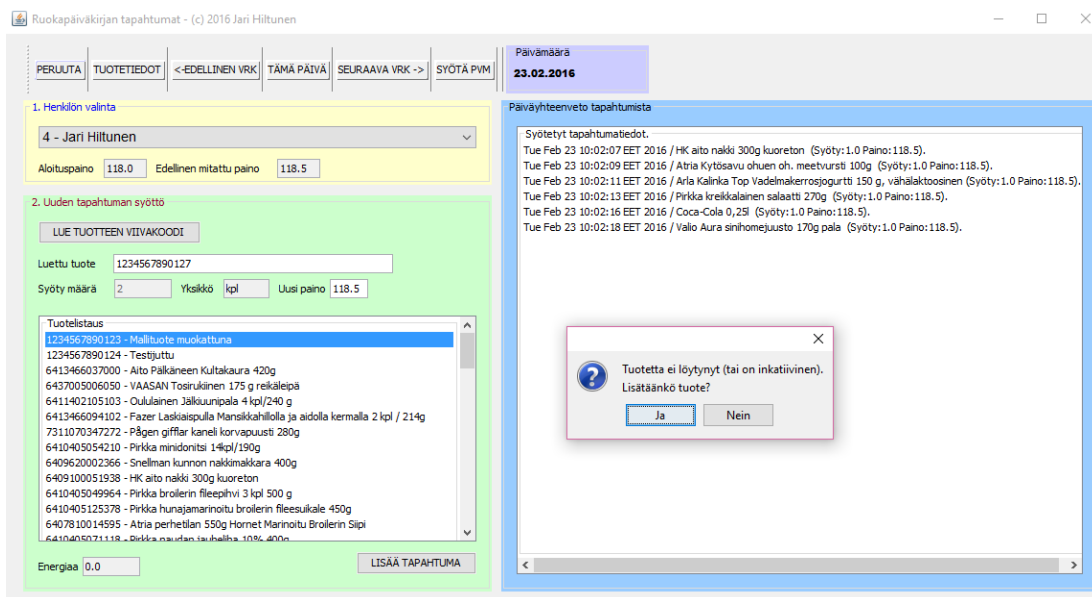
Kuva 2 Päiväkirja tuote valittuna listasta

Kysymyksen kieli voidaan muuttaa Locale.setDefault(Locale.GERMAN)- asetuksella, mutta valitettavasti en ole keksinyt miten saisin muutetuksi suomeksi. Saksa on lähinnä numeroiden käsittelyn takia.

⁸ Esimerkiksi Päiväkirjassa voit kirjoittaa päivämäärän suoraan tapahtumien yläpuolella olevaan värilliseen laatikkoon. Syöteen oikeellisuus tarkistetaan, mutta tämän jälkeen muutokset tehdään metodeilla.

⁹ Date-muotoa voidaan muuttaa locale-asetuksin.

¹⁰ Viivakoodin lukija voi olla näppäimistön rinnalle asennettava viivakoodin lukija tai mikä tahansa laite, joka osaa lukea 13-merkkistä GTIN-koodia (vaikka rfid tai kamera).



Kuva 3 Päiväkirja tuote valittuna EAN-koodilla

Tuotteita ei itse asiassa koskaan saa poistaa tietokannasta, koska silloin relaatiot menisivät rikki! Tästä syystä käyttämättömät tuotteet inaktivoidaan ja ne eivät tule esille listauksissa.

Ruokapäiväkirja.java

Ainoa luokka, jossa on main-metodi. Tästä koko sovellus lähtee käyntiin. Käytännössä ensin tarkistetaan tietokannat, luetaan ne LinkedListoihin ja sitten kännistetään Yhteenveto-luokasta löytyvä kehys.

```
/**
 * Ruokapäiväkirja-sovellus. Tämä sovellus on siirrettävissä esimerkiksi
 * mobiililaitteeseen tai esimerkiksi PDA:lle, joista viivakoodi/RFID voidaan
 * lukea käyttämällä kännykän omia ominaisuuksia. Tekele on Tietojenkäsittely-
 * tieteiden OOP:-kurssiin liittyvä harjoitus, joka oli suppeaksi rajattu.
 * Koodi on tekijänoikeudella suojattu siten, että mikäli haluat hyödyntää,
 * pyydä kirjallinen lupa ja mainitse koodissasi mistä olet kopioinut.
 * !! Tuotetietojen ja Tapahtumien välisenä relaationa toimii EAN/GTIN-koodi!!
 * !! Henkilötietojen ja Tapahtumien välisenä relattiona toimii henkilönnumero !!
 */
* Tekijä: @author Jari Hiltunen
* Versio: @version 1.0a
* SDK alkaen: @since 8.0.73
*/
import java.io.Serializable;
import java.util.*;
import java.io.*;
import java.text.ParseException;
import javax.swing.JFrame;

/** Ruokapäiväkirjan pääluokka. Tästä luokasta haaraudutaan eri ikkunoihin. */
// Jokaisessa luokassa, josta tallennetaan, tulee olla implements Serializable. */
public class Ruokapäiväkirja extends JFrame implements Serializable {
    // Globaalit linkedlistat. */
    // Taulukko tuotteille, ei saa määritellä datatyyppiä! Timantin välissä olio. */
    public static LinkedList<Tuote> tuotteet = new LinkedList<Tuote>();
    // Taulukko tapahtumille, ei saa määritellä datatyyppiä! Timantin välissä olio. */
    public static LinkedList<Tapahtuma> tapahtumat = new LinkedList<Tapahtuma>();
    // Taulukko henkilöille, ei saa määritellä datatyyppiä! Timantin välissä olio. */
    public static LinkedList<Henkilo> henkilot = new LinkedList<Henkilo>();
    // Tietokantojen nimet polkuineen, välittyy myös tarvittaviin luokkiin. */
    public static String tuotetietokanta = "tuotteet.db";
    public static String tapahtumatietokanta = "tapahtumat.db";
    public static String henkilotietokanta = "henkilot.db";

    /**
     * @param args normaalit argumentit.
     */
}
```



```
* @throws java.text.ParseException palauttaa virheen.
*/
public static void main(String[] args) throws ParseException {
    Locale.setDefault(Locale.GERMAN);
    /** Luetaan tietokannat levyltä muistiin tai luodaan tyhjät. */
    /** Tietokantojen polut ja nimet .*/

    /** Tarkistetaan onko tuotetietokanta luotu.*/
    File tuotefile = new File(tuotetietokanta);
    if(tuotefile.exists() && !tuotefile.isDirectory()) {
        /** Yritetään lukea tuotetiedot levyltä muistiin jos on. */
        tuotteet=Levytoiminta.lueTuotteet(tuotetietokanta);
    } //tiedoston tarkistus
    else { //ei ollut, luodaan tyhjä
        AlustaTietokanta.alustaTuoteTietokanta(tuotetietokanta);
    } //else
    /** Tarkistetaan onko tuotetietokanta luotu.*/
    File tapahtumafile = new File(tapahtumatietokanta);
    if(tapahtumafile.exists() && !tapahtumafile.isDirectory()) {
        /** Yritetään lukea tuotetiedot levyltä muistiin jos on. */
        tapahtumat=Levytoiminta.lueTapahtumat(tapahtumatietokanta);
    } //tiedoston tarkistus
    else { //ei ollut, luodaan tyhjä
        AlustaTietokanta.alustaTapahtumaTietokanta(tapahtumatietokanta);
    } //else
    /** Tarkistetaan onko henkilötietokanta luotu.*/
    File henkilofile = new File(henkilotietokanta);
    if(henkilofile.exists() && !henkilofile.isDirectory()) {
        /** Yritetään lukea henkilötiedot levyltä muistiin jos on. */
        henkilot=Levytoiminta.lueHenkilot(henkilotietokanta);
    } //tiedoston tarkistus
    else { //ei ollut, luodaan tyhjä
        AlustaTietokanta.alustaHenkiloTietokanta(henkilotietokanta);
    } //else
    //----- tietokannat luettu muistiin tai tehty tyhjät ----- //

    /** Käynnistetään ikkunat ja tulkitaan mahdolliset virheet käynnistyksessä. */
    try {

        javax.swing.UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
    } catch (ClassNotFoundException ex) {

        java.util.logging.Logger.getLogger(Tuotetiedot.class.getName()).log(java.util.logging.Level.SEVERE,
        null, ex);
    } catch (InstantiationException ex) {

        java.util.logging.Logger.getLogger(Tuotetiedot.class.getName()).log(java.util.logging.Level.SEVERE,
        null, ex);
    } catch (IllegalAccessException ex) {

        java.util.logging.Logger.getLogger(Tuotetiedot.class.getName()).log(java.util.logging.Level.SEVERE,
        null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {

        java.util.logging.Logger.getLogger(Tuotetiedot.class.getName()).log(java.util.logging.Level.SEVERE,
        null, ex);
    }

    /** Aloitetaan yhteenvedolla, josta haaraudutaan toisaalle. */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Yhteenvedo().setVisible(true);
        }
    });
} //main
} //luokka
```


Tapahtuma.java

Tapahtuma-luokalla muodostetaan tapahtumia tietokantaan, jotka pitävät sisällään syödyn tuotteen ja määrän lisäksi mitatun painon. Mitatuista painoista voidaan muodostaa tarvittaessa¹¹ painokäyriä yms.

Konstruktorit:

1. public Tapahtuma(Date inKelloaika, Long inGtin, Integer inHenkilo, Double inSyotyMaara, Double inPaino, String inHuomautus)
2. public Tapahtuma(Date inKelloaika, Long inGtin, Integer inHenkilo)
3. public Tapahtuma()

Tuote.java

Tuote-luokalla muodostetaan tuotetietokanta, joista voidaan valita syödyt tuotteet¹².

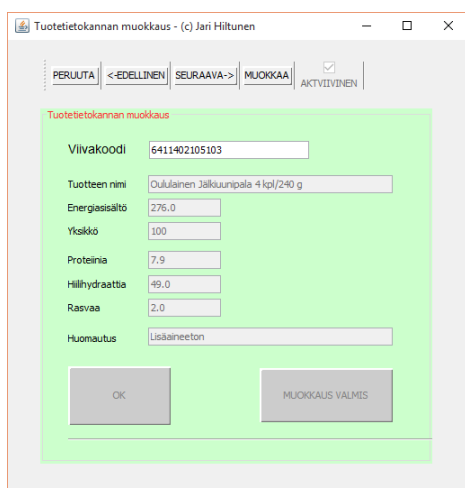
1. public Tuote(Long inGtin, String inTuoteNimi, Double inEnergia,
2. String inYksikko, Double inProteiini, Double inHiilihydraatti,
3. Double inRasva, String inHuomautus, Boolean inAktivoitu)
4. public Tuote(Long inGtin, String inTuoteNimi)
5. public Tuote()

Tuotetiedot.java

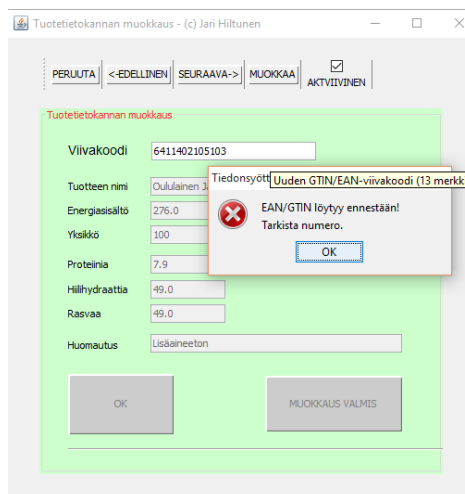
Tuotetiedot-luokalla voidaan lisätä uusia tuotteita tietokantaan tai muuttaa olemassa olevia. OK ja MUUTOS VALMIS painikkeet aktivoituvat sen mukaisesti miten valinnat on tehty. Ajatuksena on syöttää viivakoodinlukijalla tuote Viivakoodi-kenttään, jolloin sovellus tarkistaa löytyykö sitä ennestään ja jos ei löydy, kysytään tarvittavat tiedot. Tuotteita voi myös selata.

Action listenereiden (päälogiikka) lisäksi seuraavia metodeja on tehty Tuotetiedot.javaan:

1. public static String[] listaaHenkiloNimet ()
2. public static String[] listaaTuotteet ()
3. public static String[] listaaTapahtumatTanaan ()
4. public static String[] listaaTapahtumatValittu (String inPaivamaara



Kuva 4 Tarkistusesimerkki



Kuva 5 Tuotteiden selaus

¹¹ Tällä tarkoitetaan sitä, että jos haluaa jatkuvirtää sovellusta, on toiminta jo valmiiksi ajateltuna.

¹² Halutessaan voi tulostaa esimerkiksi lautasen tai listaan omia keksittyjä viivakoodeja (alkaen esim. 99999-), jolloin valmiiden omatekoisten energiamäärät saadaan kätevästi luettua.

Kuva 6 Tuotteen lisääminen

Yhteenveto.java

Yhteenveto kokoaa yhteen ja laskee tehtävän määrittäksessä olevia asioita.

Logiikka on pääosin action listenereissä. Yhteenveto pitää sisällään seuraavia metodeja:

1. public static String[] listaaHenkiloNimet ()
2. public static String[] enitenEnergiaa ()
3. public static boolean onkoListassa (LinkedList lista, String merkkijono)

Energialaskenta		Proteiinia/g	Hiiliä/g	Rasvaa/g
KCAL syöty:				
Edellinen vuorokausi	1364	58	118	70
Keskimäärin/päivä	1941	90	184	83
Kaikki yhteensä	1941	90	184	83
Ennuste-energia yhteensä	3334			

Yhteenveto	
Ensimmäinen punnituspäivämäärä	22.02.2016
Viimeisin punnituspäivämäärä	23.02.2016
Edellinen mitattu paino	70.0
Painoero (alkupaino-edellinen mittaus)	1.0
Ennustepainonalennus kg	-0,2
Ennustepaino kg	69,8

Eniten olet saanut energiaa syömällä näitä tuotteita

- 60.0 kcal / Valio pehmeä maitorahka 250 g
- 60.0 kcal / Arla 1 kg Luonto + luonnonjogurtti laktoositon
- 55.0 kcal / Kultamuna Viiräisen munia 10 kpl
- 50.0 kcal / Battery energiajuoma 0,33l
- 41.0 kcal / Dr Pepper Original 0,5l
- 300.0 kcal / K-Menu sulatejuustoviipale 600g
- 280.0 kcal / Kananpojan paneroidut Nuggetit 200g
- 254.0 kcal / RiitanHerku Super kinkku-metwursti pizza 250g
- 140.0 kcal / Pirkka lihamakaronilaatikko 400 g
- 130.0 kcal / Fresh Loupas Hatki Kanapaasar salaatti 210g

Kuva 7 Yhteenvetonäkymä