JAVA CLI -HARJOITUKSET

Tässä dokumentissa olevat Java-harjoitukset liittyvät CLI-ohjelmiin, mikä tarkoittaa että ohjelmissa on Command-Line User Interface, eli niitä voidaan suorittaa esim. Windowsin Command Prompt -ikkunassa.

Näiden harjoitusten tarkoitus on opiskella Java-kielen perusasioita ja kieleen kuuluvia yleiskäyttöisiä standardiluokkia ennenkuin siirrytään esim. graafisten käyttöliittymien tekemiseen.

Kari Laitinen
http://www.naturalprogramming.com
2015-01-04 Tiedosto luotu.
2015-01-13 Viimeisin muutos

OHJEITA HARJOITUSTEN TEKOON

- Nämä harjoitukset ovat pääsääntöisesti sellaisia, että alussa otetaan käyttöön jokin esimerkkiohjelma, jota muutetaan ja edelleenkehitetään harjoituksissa. Jokaiseen harjoitukseen kuuluu yleensä monta osatehtävää. Tarkoitus on että yhdellä harjoituskerralla tehdään aina johonkin yhteen ohjelmaan liittyvät tehtävät.
- JCreatorilla työskenneltäessä ei ole tarpeellista perustaa projekteja. Riittää kun .java-tiedostot avataan sellaisenaan JCreatoriin ja tarvittavat tiedostot ovat kaikki samassa kansiossa.
- Varo antamasta Java-ohjelmallesi esim. nimeä String.java, koska String on Javan standardiluokka. Anna ohjelmallesi mieluiten pitkähkö nimi tyyliin KivaPeliHarjoitus.java. Muista myös että Javassa on sääntö jonka mukaan ohjelman pääluokan nimi tulee olla KivaPeliHarjoitus jos tiedostonimi on KivaPeliHarjoitus.java.
- Säädä ohjelmaeditorisi sellaiseksi että tabulointinäppäimen painallus vastaa kolmea välilyöntiä. Tällaista ohjelmointityyliä on käytetty Kari Laitisen esimerkkiohjelmissa joita näissä harjoituksissa muutellaan. Paras on säätää editori sellaiseksi että tabulointimerkin tilalle tiedostoon tulee kolme välilyöntiä. (Yleensä editoreissa on Settings- tai Configure-menu josta tämmöisen säätämisen voi tehdä.)

Java-harjoitus: Lämpötiloja muuntava ohjelma

Maailmalla on kaksi yleisesti käytettyä tapaa lämpötilojen esittämiseen. Fahrenheit-asteet (ºF) ovat käytössä USA:ssa ja joissain muissa valtioissa, kun taas Celsius-asteet (ºC) ovat käytössä useimmissa Euroopan maissa ja monissa muissa maissa maapallolla. Veden jäätymispiste on 0 Celsius-astetta ja 32 Fahrenheit-astetta, 10ºC on 50ºF, 20ºC on 68ºF, 30ºC on 86ºF, ja niin edelleen. 10 Celsius-astetta siis vastaa 18-astetta Fahrenheit-asteikossa.

1. Kirjoita ohjelma (esim. Lampo.java) joka osaa muuntaan annetut Fahrenheit-asteet Celsius-asteiksi. Voit halutessasi tehdä myös toiseen suuntaan muuntavan ohjelman.

2. Paranna ohjelmaa siten että se muuntaa sille syötetyt asteet sekä Celsius-asteiksi että Fahrenheit-asteiksi. Ohjelma siis ottaa sille syötetyn lämpötila-arvon ja tekee automaattisesti muunnoksen molempiin suuntiin.

3. Paranna ohjelmaa lisäämällä siihen ominaisuus, että jos käyttäjä syöttää lämpötilaksi arvon nolla, niin se tulostaa taulukon jossa Fahrenheit-asteet muutetaan 10 asteen välein Celsius-asteiksi. Tähän ohjelmaan tarvitset if-rakenteen, jolla tutkit onko annettu arvo nolla. Lisäksi tarvitset silmukan joka tulostaa lämpötilataulukon. (Esim. ohjelmassa SuurinLuku.java on käytössä if-rakenteita ja mm. ohjelmassa Summaussilmukka.java on esimerkki while-silmukan käytöstä. Ohjelmia Etaisyys.java ja Formatointeja.java tutkimalla saat selville kuinka tulostus voidaan formatoida printf()-metodia käytettäessä.)

4. Paranna ohjelmaa vielä siten että lämpötilataulukon tulostuksen suorittaa erillinen kutsuttava metodi. Metodi voi alkaa esim. seuraavasti

```
public static void tulosta_lampotilataulukko()
{
   ...
}
```

ja sitä voidaan kutsua main-metodista seuraavaan tapaan

```
if ( annettu_lampotila  ==  0 )
{
   tulosta_lampotilataulukko() ;
}
```

Esim. ohjelmasta Kirjaimet.java näet kuinka parametritonta metodia voidaan kutsua.

5. Jos intoa piisaa, paranna ohjelmaa siten että sillä saa tulostettua lämpötilataulukon tiedostoon. Ohjelmasta http://www.naturalprogramming.com/javaohjelmat/javaohjelmat3/KopioiTiedosto.java näet kuinka tekstitiedostoon voidaan kirjoittaa tekstiä.

Java-harjoitus: SILMUKKA JOKA TULOSTAA KERTOTAULUN

1. Tee ohjelma "Kerro.java" joka tulostaa 4:n kertotaulun while- tai for-silmukassa tyyliin

```
 1 * 4  =   4
 2 * 4  =   8
 3 * 4  =  12
 4 * 4  =  16
       ...
 9 * 4  =  36
10 * 4  =  40
```

Ohjelmassa tulee siis numeroarvojen (muuttujien) väliin saada tulostettua tekstit " * " ja " = ". Muista että kertolaskuoperaattori on Javassa * kuten C++:ssakin.

2. Saatuasi aikaan 4:n kertotaulun tulostuksen, voit parantaa ohjelmaa vielä siten että se kysyy käyttäjältä, minkä luvun kertotaulu tulostetaan. Kuten esim. ohjelmasta Peli.java voit nähdä, voidaan int-tyypin luku lukea näppäimistöltä seuraavasti

```
kerrottava  =  nappaimisto.nextInt() ;
```

Jotta tämä lause on mahdollinen, on näppäimistöolio määriteltävä ensin main()-metodin alussa ja ohjelmassa on oltava alussa lause import java.util.* ;

3. Paranna ohjelmaa siten että ohjelma tulostaa myös annettujen lukujen jakolasku- ja jakojäännösoperaatiot kertotaulun sivuun

```
1 * 4  =   4     1 / 4  =  0    1 % 4  =  1
2 * 4  =   8     2 / 4  =  0    2 % 4  =  2
3 * 4  =  12     3 / 4  =  0    3 % 4  =  3
4 * 4  =  16     4 / 4  =  1    4 % 4  =  0
5 * 4  =  20     5 / 4  =  1    5 % 4  =  1
  ... jne
```

4. Muuta ohjelmaa vielä siten että aritmeettiset tulostukset tehdään erillisellä staattisella metodilla jota kutsutaan main() -metodista sen jälkeen kun käyttäjä on antanut näppäimistöltä luvun. Tulostuksen suorittava metodi kirjoitetaan luokan Kerro sisään esim. seuraavaan tapaan

```
public static void tulosta_kertotaulu_yms(
                                        int  kerrottava )
{
   ...
}
```

Esim. ohjelmasta Suurin.java näet kuinka erillistä metodia kutsutaan

5. Lisää ohjelmaan vielä metodi

```
public static void tulosta_kertotaulu_yms_heksana(
                                          int  kerrottava )
{
   ...
}
```

Tämän metodin saa helposti aikaiseksi kun kopioi edellisessä kohdassa tehdyn metodin ja muuttaa sitä sopivasti tulostuksen osalta. Luvun saa tulostettua heksadesimaalisena käyttämällä Integer -luokan metodia toHexString() seuraavaan tapaan

```
System.out.print( "joku_luku heksana on  "
                +   Integer.toHexString( joku_luku ) ) ;
```

Vaihtoehtoisesti luku voidaan tulostaa heksadesimaalisena käyttämällä metodia System.out.printf() ja formatointimäärittelyä "%X".

HARJOITUKSIA OHJELMALLA Elaimia.java

Harjoitus 1:

Lisää luokkaan Elain metodi tyhjenna_vatsa(), jonka avulla Elain-olion vatsa tyhjennetään siten että sinne kirjoitetaan tyhjä stringi "". Metodia tulee voida kutsua esim. seuraavasti

```
kissaolio.tyhjenna_vatsa() ;
koiraolio.tyhjenna_vatsa() ;
```

Tämän muutoksen onnistumisen voi testata kutsumalla anna_puhua() -metodia ja tutkimalla onko vatsa todella tyhjentynyt.

Harjoitus 2:

Lisää luokkaan Elain uusi datakenttä

```
String  elaimen_nimi ;
```

Tässä on muutettava luokan ensimmäistä konstruktoria siten että Elain-olio voidaan luoda esim. lauseella

```
Elain nimetty_kissa  =  new Elain( "kissa", "Miuku" ) ;
```

Myöskin kopiointikonstruktoria on muutettava siten että uusi datakenttä tulee kopioiduksi. Metodia anna_puhua() tulee modifioida siten että se tekee seuraavantapaisen tulostuksen

```
Hei. Mina olen kissa nimelta Miuku.
Olen syonyt: ...
```

Harjoitus 3:

Muuta metodia anna_puhua() siten että se tulostaa

```
Hei. Mina olen ... nimelta ...
Vatsani on tyhja.
```

siinä tapauksessa kun vatsan_sisalto viittaa tyhjään stringiin. Vatsa on tyhjä niin kauan kuin metodia ruoki() ei ole kutsutta. Voit käyttää String-luokan metodia length() tarkistamaan onko vatsa tyhjä. Tätä metodia voi käyttää esimerkiksi seuraavaan tapaan

```
if ( vatsan_sisalto.length()  ==  0 )
{
   // vatsan_sisalto viittaa tyhjaan stringiin.
   ...
```

Jos vatsa ei ole tyhjä, annetaan alkuperäisen kaltainen tulostus.

Harjoitus 4:

Lisää luokkaan Elain oletuskonstruktori eli konstruktori jota voidaan kutsua antamatta parametreja (argumentteja). Tällä konstruktorilla Elain-tyypin olio voidaan luoda esim. seuraavasti

```
Elain joku_elain  =  new Elain() ;
```

Oletuskonstruktori voi asettaa datakentän lajin_nimi arvoksi stringin "oletuselain" ja datakentän elaimen_nimi arvoksi "nimeton elain". Toisin sanoen, kun anna_puhua()-metodia kutsutaan oletuskonstruktorilla luodulle Elain-oliolle, syntyy seurvaavankaltainen tulostus

```
Hei. Mina olen oletuselain nimelta nimeton elain.
...
```

Luokassa on hyvä olla oletuskonstruktori siinä tapauksessa kun siitä johdetaan (periytetään) uusia luokkia. Kun periytetyn luokan olio luodaan, useimmissa tapauksissa kantaluokan oletuskonstruktoria kutsutaan automaattisesti ennenkuin periytetyn luokan konstruktori suoritetaan.

Harjoitus 5:

Kirjoita ohjelmaan, esimerkiksi luokan Elain jälkeen, uusi luokka nimeltä Elaintarha. Elaintarha-luokan on tarkoitus edustaa olioita jotka sisältävät joukon Elain-olioita. Tässä luokassa ei välttämättä tarvita konstruktoria kun datakentät alustetaan niiden esittelyn yhteydessä. Luokassa Elaintarha tulee olla metodi nimeltä lisaa_elain(), jolla uusi Elain-tyyppinen olio voidaan laittaa osaksi eläintarhaa. Lisäksi siinä tulee olla metodi anna_elainten_puhua(), jonka sisällä Elain-luokan anna_puhua()-metodia kutsutaan jokaisen tarhan eläimen suhteen. Elaintarha-luokka voi näyttää seuraavanlaiselta

```
class  Elaintarha
{
   Elain[]  tarhan_elaimet  =  new  Elain[ 20 ] ;

   int  elaimia_tarhassa  =  0 ;

   public void lisaa_elain( Elain  uusi_elain_tarhaan )
   {
      ...

   public void anna_elainten_puhua()
   {
      for ( int elaimen_indeksi  =  0 ;
                elaimen_indeksi  <  elaimia_tarhassa ;
                elaimen_indeksi  ++ )
      {
         ...
```

Tarkoitus on että Elaintarha-luokka sisältää Elain[]-tyyppisen taulukon joka sisältää viittaukset tarhan Elain-olioihin. Muuttujalla elaimia_tarhassa pidetään kirjaa siitä montako eläintä tarhassa on. Ohjelmassa Olympialaiset.java on esimerkki olioviittauksia sisältävän taulukon käytöstä.

Metodissa main() voidaan uutta Elaintarha-luokkaa testata esimerkiksi seuraavanlaisilla lauseilla:

```
Elaintarha  testitarha  =  new  Elaintarha() ;

testitarha.lisaa_elain( kissaolio ) ;
testitarha.lisaa_elain( koiraolio ) ;
testitarha.lisaa_elain( toinen_kissa ) ;
testitarha.lisaa_elain( joku_elain ) ;

testitarha.anna_elainten_puhua() ;
```

Harjoitus 6: (vanha harjoitus)

Muuta Elain-luokan datajäsen vatsan_sisalto String-olioita sisältäväksi taulukoksi joka määritellään luokan alussa seuraavaan tapaan

```
String[] vatsan_sisalto  =  new  String[ 30 ] ;
```

Kun vatsan_sisalto määritellään kuten yllä, siinä on tilaa 30 ruokastringille. Tarkoitus on että eläinoliota ruokittaessa ruokana annettava stringi lisätään tämän taulukon ensimmäiseen vapaaseen paikkaan. Ensimmäisellä ruokintakerralla ruokastringi lisätään taulukon ensimmäiseen paikkaan. Jotta taulukkoa voidaan käyttää, tulee luokkaan määritellä myös datajäsen

```
int ruokintojen_maara  =  0 ;
```

jota voidaan käyttää taulukon indeksinä. Tämä muutos vaatii muutoksia konstruktoreihin ja muihin luokan metodeihin. Metodeiden "signeerauksia", siis niiden ottamien parametrien (argumenttien) tyyppejä, ei tarvitse muuttaa. Näin myöskään metodia main() ei tarvitse tässä kohdassa muuttaa.

Esimerkiksi metodissa ruoki() tulee tehdä sellainen muutos että stringi annettu_ruoka kopioidaan taulukkoon vatsan_sisalto.
Tämä voi tapahtua seuraavasti

```
vatsan_sisalto[ ruokintojen_maara ]  =  annettu_ruoka ;
```

Metodi anna_puhua() voi tutkia vatsan sisällön tyhjyyttä käyttämällä hyväksi datajäsentä ruokintojen_maara. Vatsan sisältö tulee tulostaa silmukalla seuraavaan tapaan

```
for ( int ruoan_indeksi  =  0 ;
          ruoan_indeksi  <  ruokintojen_maara ;
          ruoan_indeksi  ++ )
{
   System.out.print( ...
           // tulostetaan yksi syöty ruoka kerrallaan
```

Harjoitus 7: (vanha harjoitus)

Johda luokasta Elain perinnan avulla uusi luokka Petoelain, ja kirjoita tähän uuteen luokkaan uusi versio metodista ruoki(). Tämä uusi metodi ruoki() tulee olla sellainen että Petoelain-oliolle voidaan syöttää toisia eläimiä. Tämä uusi ruoki()-metodi voi alkaa seuraavasti:

```
public void ruoki( Elain syotava_elain )
{
```

Toisen eläimen syominen voi tapahtua esimerkiksi siten että syötävän eläinparan datajäsen lajin_nimi kulkeutuu Petoelain-olion vatsaan. Uudessa ruoki() -metodissa voidaan viitata argumenttina (parametrina) tulevan Elain-olion datajäseneen lajin_nimi kirjoittamalla

```
syotava_elain.lajin_nimi
```

Syötävän eläimen datajäsen lajin_nimi voidaan kopioida Petoelain-olion vatsaan seuraavanlaisella käskylauseella

```
vatsan_sisalto[ ruokintojen_maara ]  =
                                syotava_elain.lajin_nimi ;
```

Uusi Petoelain-luokka kannattaa rakentaa siten että tekee perinnän avulla ensin uuden luokan, eikä lisää siihen muita metodeita kuin konstruktorin. Tämän jälkeen voi testata että uuden Petoelain-luokan olioita voi luoda, ja niille voi kutsua vanhoja metodeita. Vasta tämän jälkeen kannattaa tehdä uusi versio ruoki() -metodista. Katso mallia perinnän toteuttamiseen esimerkiksi ohjelmasta PankkiMonimuotoinen.java

Uusi luokka Petoelain voidaan kirjoittaa luokan Elain jälkeen jo olemassaolevaan ohjelmatiedostoon. (Useita luokkia voi olla kirjoitettuna samaan .java-tiedostoon jos niitä ei ole määritelty public-luokiksi.)

Petoelain-luokan toimintaa voidaan testata seuraavanlaisilla käskylauseilla:

```
Petoelain  tiikeri  =  new Petoelain( "..." ) ;
Elain      nauta    =  new Elain( "..." ) ;

tiikeri.ruoki( nauta ) ;
```

HARJOITUKSIA OHJELMALLA Olympialaiset.java

Harjoitus 1:

Nykyisellään ohjelma ei tiedä että vuonna 2012 olympialaiset pidetään Lontoossa Iso-Britanniassa. Korjaa ohjelmaa siten että myös noiden tulevien vuoden 2012 olympialaisten tiedot tulostuvat.

Harjoitus 2:

Nykyisellään ohjelmassa datan loppu kisataulukossa on merkitty "olympialaisilla" joiden olympiavuosi on 9999. Taulukosta tietoa hakeva algoritmi tunnistaa olympiadatan lopun tuon erikoisen olympiavuoden avulla. Toinen mahdollisuus tunnistaa datan loppu olioviittauksia sisältävästä taulukosta on etsiä milloin taulukosta löytyy null, eli tieto siitä että kyseisestä taulukon muistipaikasta ei viitata mihinkään olioon. Olioviittauksia sisältävä taulukko sisältää null-arvoja (nollia) heti taulukon luomisen jälkeen. Kun taulukon muistipaikasta ryhdytään viittaamaan johonkin olioon, ko. muistipaikka saa jonkin muun arvon kuin null. Muuta ohjelma sellaiseksi että datan loppu tunnistetaan null-viittausta etsimällä. Jotta null voidaan tunnistaa, on dataa hakevassa if-rakenteessa testattava ensin kohdattiinko taulukon loppu, ja vasta tämän jälkeen testataan onko taulukon kautta löydetyn olion olympiavuosi sama kuin annettu vuosi. if-rakenteen tulee siis alkaa seuraavasti

```
if ( kisataulukko[ kisojen_indeksi ]  ==  null )
{
   //  Data loppui taulukosta.
```

Tämän muutoksen ansiosta ohjelman toiminta ei saa muuttua. Tässä tehdään ohjelman rakenteesta hiukan järkevämpi.

Harjoitus 3:

Johda luokasta Olympialaiset uusi luokka nimeltä Talviolympialaiset. Voit kirjoittaa Talviolympialaiset-luokan heti Olympialaiset-luokan perään samaan tiedostoon. Jotta luokka Olympialaiset voi toimia "tehokkaasti" kantaluokkana toiselle luokalle on sinne lisättävä seuraavanlainen oletuskonstruktori

```
public Olympialaiset() {} // Tyhja oletuskonstruktori
```

Oletuskonstruktori tarvitaan koska se suoritetaan automaattisesti ennen johdetun luokan konstruktorin suorittamista.

Tee aluksi Talviolympialaiset-luokalle kostruktori joka on samanlainen kuin Olympialaiset-luokan konstruktori. Muista toki että Talviolympialaiset-luokan konstruktorin nimi tulee olla Talviolympialaiset.

Kun Talviolympialaiset-luokalla on konstruktori, voit kokeilla Talviolympialaiset-olion luontia ja taulukkoon tallettamista seuraavanlaisella lauseella

```
kisataulukko[ 28 ] = new Talviolympialaiset( 2006,
                                    "Torino", "Italia" ) ;
```

Jos ohjelmasi löytää Talviolympialaiset-olion tiedot, olet suorittanut tämän harjoituksen.

Harjoitus 4:

Edellisessä harjoituksessa tehty Talviolympialaiset-luokka ei eroa toiminnaltaan Olympialaiset-luokasta. Paranna Talviolympialaiset-luokkaa lisäämällä siihen uusi versio metodista tulosta_olympialaisten_tiedot() siten että metodin tulostamassa tekstissä mainitaan sana "talvi" esim. seuraavaan tapaan

```
Vuonna 2006 talviolympialaisten kaupunki oli Torino (Ita-
lia).
```

Tarkoitus on että Talviolympialaiset-luokassa ylikirjoitetaan luokassa Olympialaiset määritelty metodi. Kun tämmöinen metodi on olemassa, sitä käytetään automaattisesti silloin kun kisataulukosta löydetään Talviolympialaiset olio.

Harjoitus 5:

Aikaisempina vuosina talvi- ja kesäolympialaiset olivat samana vuonna. Esim. vuonna 1984 talviolympialaiset olivat Sarajevossa Jugoslaviassa. Jos tämmöisiä vanhojen talviolympialaisten tietoja lisätään kisataulukkoon, syntyy ongelma että ohjelma ei löydä niitä koska samana vuonna olleiden kesäkisojen tiedot tulostuvat ensin, ja tietojen etsintä lopetetaan tämän jälkeen. Muuta etsintäalgoritmia siten että etsintää jatketaan aina taulukon loppuun, siis ensimmäiseen null-olioviittaukseen saakka, ja tulostetaan kaikkien annettuna vuonna olleiden olympialaisten tiedot. Ohjelmaan tarvitaan todennäköisesti uusi boolean-tyypin muuttuja kuten

```
boolean olympiatietoja_loydetty  =  false ;
```

jolle annetaan arvo true silloin kun taulukosta löydetään annetun vuoden olympiatietoja. Jos tämä muuttuja on lopussa false, annetulle vuodelle ei löydetty olympiatietoja.

Harjoitus 6:

Muuta ohjelmaa siten että se tulostaa kaikki kisataulukossa olevat kesäolympialaisten tiedot siinä tapauksessa että sille annetaan vuodeksi nolla, tai jos sille annetaan vuodeksi 1, se tulostaa kaikkien talviolympialaisten tiedot. Tarkoitus on että ratkaiset tämän tehtävän oliosuuntautuneesti siten että käytät Javan instanceof-operaattoria. Tällä operaattorilla voidaan tutkia, onko taulukosta viitattu olio jotain tiettyä tyyppiä. instanceof-operaattoria voidaan käyttää seuraavaan tapaan

```
if ( annettu_vuosi  ==  1 )
{
   kisojen_indeksi  =  0 ;

   while ( kisataulukko[ kisojen_indeksi ]  !=  null )
   {
      if ( kisataulukko[ kisojen_indeksi ] instanceof
                                          Talviolympialaiset )
      {
         // Viitataan Talviolympialaiset-olioon.
```

instanceof palauttaa arvon true jos sen vasemmalla puolella viitataan olioon joka on oikealla puolella annetun luokan tai sen alaluokan olio. Jotta voit löytää instanceof-operaattorilla myös kesäkisojen tiedot, tulee sinun johtaa luokasta Olympialaiset luokka Kesaolympialaiset aivan samaan tapaan kuin siitä on jo johdettu luokka Talviolympialaiset. Kisataulukkoon talletettavat Olympialaiset-oliot muutat sitten (Find/Replace-toiminnolla) Kesaolympialaiset-olioiksi. Taulukon itsensä tyyppi pitää edelleen olla Olympialaiset.

EDITORIN RAKENTAMISHARJOITUKSET

Näissä harjoituksissa rakennamme yksinkertaisen tekstieditorin jonka avulla komentorivi-ikkunassa kirjoitettu teksti menee tekstitiedostoon kiintolevylle. Tässä tullaan siis kokeilleeksi tiedostoon kirjoittamista ja valmiiden tiedostoluokkien käyttöä. Lisäksi harjoittelemme `ArrayList`-pohjaisen taulukon käyttöä.

Harjoitus 1:

Tee aluksi ohjelma (esim. **Editor.java**) joka lukee näppäimistöltä annettuja tekstirivejä ja tallettaa annetut tekstirivit `ArrayList<String>`-tyyppiseen taulukkoon. Ohjelman tulee toimia siten sille voidaan syöttää myös tyhjiä tekstirivejä. Kun ohjelma saa rivin jonka ensimmäinen merkki on piste '.', se lopettaa tekstin lukemisen näppäimistöltä ja tallettaa tekstirivit tiedostoon.

`ArrayList`-luokkaan perustuva taulukko luodaan esim. seuraavasti

```
ArrayList<String> given_text_lines =
                                  new ArrayList<String>() ;
```

Tämä lause luo `ArrayList`-taulukon johon voidaan tallettaa tyyppiä `String` olevia olioita, eli esim. tekstirivejä. `ArrayList`-luokan käyttämiseksi täytyy ohjelmassa olla esim. seuraava import-komento:

```
import java.util.* ;  // ArrayList, Scanner, etc.
```

Silmukka, joka lukee tekstirivejä näppäimistöltä voi olla seuraavanlainen

```
boolean user_wants_to_type_more = true ;

while ( user_wants_to_type_more == true )
{
   String text_line_from_user = keyboard.nextLine() ;

   if ( text_line_from_user.length() > 0 &&
        text_line_from_user.charAt( 0 ) == '.' )
   {
      user_wants_to_type_more = false ;
   }
   else
   {
      // Here you should add the string to the
      // ArrayList array.
   }
}
```

`ArrayList`-luokassa on metodi `add()` jolla tekstirivi voidaan tallettaa taulukkoon.

Ennenkuin kokeilet tallettaa annettuja tekstirivejä tiedostoon, kannattaa varmistua että rivit ovat todella tallettuneet `ArrayList`-pohjaiseen taulukkoon. Tämän varmistamiseksi voit

tulostaa kaikki tekstirivit näytölle seuraavanlaisen ’foreach’-silmukan avulla:

```
System.out.print( "\nGIVEN LINES: \n" ) ;

for ( String text_line : given_text_lines )
{
   System.out.print( "\n" + text_line ) ;
}
```

Harjoitus 2:

Kun olet saanut varmistettua että tekstirivit tallentuvat `ArrayList`-taulukkoon, paranna ohjelmaa siten että tekstirivit talletetaan tiedostoon. Tee ohjelmasta aluksi sellainen että se pistää annetut tekstirivit aina samannimiseen tekstitiedostoon (esim. **tekstia.txt**)

Kun katsot mallia ohjelmasta **FindReplace.java**, näet kuinka `ArrayList`-taulukossa olevat tekstirivit voidaan tallettaa tiedostoon. Itse asiassa tuossa ohjelmassa on valmis metodi jolla saat sitten tekstin talletettua `ArrayList`-taulukosta tiedostoon. Tiedostonkäsittelyyn liittyvien luokkien käyttämiseksi tarvitset ohjelmaan esim. seuraavan import-komennon:

```
import java.io.* ;    // Classes for file handling.
```

Varmistuaksesi että tekstirivit ovat todella tallettuneet tiedostoon, voit käyttää seuraavaa komentoa Command Prompt -ikkunassa:

```
type tekstia.txt
```

Harjoitus 3:

Paranna editoria siten että se pyytää käyttäjältä tiedostonimen siinä vaiheessa kun tekstiä aletaan tallettaa tiedostoon. Tarkoitus on siis että editori ei tallenna tekstirivejä aina samannimiseen tiedostoon.

Harjoitus 4:

Paranna editoriohjelmaa siten että se varmistaa seuraavat asiat ennenkuin tiedostonimi hyväksytään:

- Jos annetulla nimellä on jo olemassa tiedosto, ohjelma pyytää uuden tiedostonimen.
- Ohjelma ei hyväksy tyhjää stringiä tiedostonimeksi.
- Ohjelma hyväksyy vain tiedostonimet joiden nimien laajenteena on **.txt**.

Nämä testit voidaan tehdä esim. seuraavantapaisella silmukalla (jossa jo kaksi ensimmäistä testiä on toteutettu). Luokassa `String` on metodi nimeltä `endsWith()` jolla voidaan tutkia päättyykö jokin stringi johonkin tiettyyn stringiin.

```
String given_file_name = "notknown.txt" ;

boolean acceptable_file_name_given = false ;

while ( acceptable_file_name_given == false )
{
   System.out.print(
            "\nGIVE FILE NAME TO STORE THIS TEXT: " ) ;

   given_file_name = keyboard.nextLine() ;

   if ( new File( given_file_name ).exists() )
   {
      System.out.print( "\nThis file already exists!" ) ;
   }
   else if ( given_file_name.length() == 0 )
   {
      ...
```

Harjoitus 5:

Paranna editoria siten että jos käyttäjän antamalla nimellä on jo tiedosto olemassa, editori kysyy saako ko. tiedoston ylikirjoittaa.

HARJOITUKSIA OHJELMALLA BankPolymorphic.java

Näissä harjoituksissa on aiheena `ArrayList`-taulukoiden käyttö ja `Comparable`-rajapinnan toteutus luokkaan.

Näiden harjoitusten tekemisessä voi ottaa mallia ohjelmasta **HistoricalEvents.java**

Harjoitus 1:

BankPolymorphic.java on ohjelma jossa luodaan erilaisia kuvitteellisia pankkitilioloita ja kutsutaan metodeita ko. olioille.

Tutustu ensin ohjelmaan ja laita sitten `main()`-metodiin `ArrayList<BankAccount>` -tyyppinen taulukko ja talleta ohjelmassa luodut pankkitilioliot kyseiseen taulukkoon. Tee ohjelmaan pari uutta pankkitilioliota ja lisää nekin ko. taulukkoon. Tarvittavan taulukon voit luoda seuraavanlaisella lauseella:

```
ArrayList<BankAccount> bank_accounts = new ArrayList<BankAccount>() ;
```

Kun taulukko on luotu tällaisella lauseella, siihen voidaan tallettaa luokan `BankAccount` olioita sekä luokan `BankAccount` alaluokkien olioita.

Kun olet lisännyt pankkitilioliot tekemääsi taulukkoon, käy taulukko läpi ’foreach’-silmukalla ja kutsu kaikille olioille `print_account_data()`-metodia.

Harjoitus 2:

Modifioi luokkaa `BankAccount` siten että laitat sen toteuttamaan `Comparable`-rajapinnan. Tämä tarkoittaa mm. että ko. luokkaan pitää laitaa `compareTo()`-metodi joka määrittelee miten pankkitilejä vertaillaan. Voit toteutaa vertailun siten että pankkitilien suuruusjärjestyksen määrää tilin saldo eli datajäsen `account_balance`.

Kun olet saanut `Comparable`-rajapinnan toteutettua, voit testata sitä sitten niin että laitat `ArrayList`-pohjaisen taulukon järjestykseen `Collections.sort()`-metodikutsulla. Voit tulostaa pankkitilit ennen ja jälkeen sorttauksen, jolloin näet onnistuuko sorttaaminen.

Kun olet saanut tilin saldon mukaisen sorttauksen toimintaan, voit kokeilla vielä sitä että muutat `compareTo()`-metodia siten että se laittaakin pankkitilit järjestykseen omistajan nimen mukaan. Tämä on helppo toteutaa koska standardiluokka `String` toteuttaa `Comparable`-rajapinnan.

Harjoitus 3:

Johda (periytä) luokasta `BankAccount` uusi luokka nimeltä `RichPersonAccount`, johon laitat uuden datajäsenen:

```
protected double  minimum_amount_to_deposit ;
```

Tarkoitus on että tämän luokan pankkitilioliot ovat sellaisia että sinne ei saa pieniä summia tallettaa, vaan vähimmäistalletuksen määrää yo. datajäsen. Tähän uuteen luokkaan täytyy uudelleenkirjoittaa metodi `deposit_money()` joka siis hyväksyy vain talletukset jotka ovat vähintään tuon datajäsenen suuruisia. Kun testaat uuden luokan toimintaa huomaat että

yläluokkaan määritelty `Comparable`-rajapinta toimii siinäkin automaattisesti.

REGULAR EXPRESSIONS -HARJOITUKSET

Näissä harjoituksissa on aiheena regular expressions eli säännölliset lausekkeet, joiden avulla voidaan merkkijonoista etsiä haluttuja säännönmukaisia tekstiosia.

Harjoitusten päämääränä on tehdä ohjelma jolla voidaan ohjelmatiedostossa (esim. .java-tiedostossa) olevat nimet muuttaa siten, että jos siellä on käytetty alaviivallisia nimiä niin ne muutetaan ns. camel case -nimiksi. Tämä tarkoittaa että jos ohjelmassa on esim. muuttuja

```
int joku_muuttujan_nimi ;
```

niin se muutetaan muotoon

```
int jokuMuuttujanNimi ;
```

Alaviivallisia nimiä sanotaan Wikipedian mukaan snake case -nimiksi, ja nimiä joissa yksittäiset sanat on eroteltu isolla alkukirjaimella sanotaan camel case -nimiksi.

Harjoitus 1:

Ratkaise aluksi ongelma: Miten yksittäinen snake case -nimi muutetaan camel case -nimeksi?

Tämän ratkaisemiseksi voit ottaa käyttöön ohjelman **SplittingAtoms.java** jossa näytetään miten yksittäinen stringi pilkotaan jonkin merkin kohdalta siten että muodostuu stringitaulukko pilkotun stringin osista. Voit muuttaa **SplittingAtoms.java**-ohjelmaa siten

että jos sen alkuun määritellään

```
String name_with_underscores = "these_are_words_in_name" ;
```

se osaa muuntaa tämän stringin muotoon `theseAreWordsInName`. Huomaa että nimen ensimmäistä sanaa ei aloiteta isolla alkukirjaimella. Esimerkiksi ohjelmasta **Capitals.java** löydät valmiin lausekkeen jolla sanan 1. kirjain saadaan muutettua isoksi kirjaimeksi.

Kun olet ratkaissut tämän osatehtävän, on helpompi ryhtyä miettimään ongelmaa laajemmin.

Harjoitus 2:

Tämän harjoituksen ongelma voidaan esittää kysymyksenä: Miten ohjelmakoodista etsitään nimet joissa on sanojen erottimina käytetty alaviivaa?

Vastaus tähän löytyy kun otat käyttöön ohjelman **NamesFromSourceFile.java** ja muokkaat sen itsellesi sopivaksi. Nykyisellään ohjelma etsii kaikki nimet ohjelmasta. Sinun tulee muokata ohjelmakoodia niin että otat käyttöön sellaisen säännöllisen lausekkeen joka etsii nimenomaan alaviivalliset nimet.

Huomaa että tässä sinulla pitää olla jokin testitiedosto jossa on alaviivallisia nimiä. Kannattaa ottaa käyttöön jokin Kari Laitisen lyhyehkö esimerkkiohjelma ja tallettaa se esim. nimellä **Testfile.java**.

Harjoitus 3:

Tässä osatehtävässä voit jatkaa edellisessä harjoituksessa tehtyä ohjelmaa. Tarkoitus on saada aikaan löydetyille alaviivallisille nimille korvaajat

Edellisen harjoituksen seurauksena alaviivalliset nimet ovat `HashSet`-tyyppisessä tietorakenteessa johon viitataan nimellä `found_names`. Tässä voit menetellä siten että teet Harjoitus 1:ssä kehitetyllä tavalla näille nimille korvaajat. Aluksi kannattaa kuitenkin muuttaa `HashSet`-tietorakenne tavalliseksi `String[]` -taulukoksi seuraavaan tapaan:

```
String[] found_names_array =  found_names.toArray( new String[ 0 ] ) ;
```

Lisäksi kannattaa luoda taulukko johon laitetaan nimien korvaajat:

```
String[] name_replacements = new String[ found_names_array.length ] ;
```

Näiden lauseiden avulla sinulla on käytössä kaksi `String[]`-taulukkoa jotka ovat saman kokoisia. Nyt pitää jälkimmäiseen taulukkoon saada aikaiseksi ne korvaavat nimet. Ne voidaan tehdä seuraavan kaltaisella silmukalla:

```
for ( int name_index = 0 ;
          name_index < found_names_array.length ;
          name_index ++ )
{
   String  name_with_underscores  = found_names_array[ name_index ] ;



   // Tähän väliin pitäisi sitten mahduttaa se mitä kehitit
   // Harjoitus 1:ssä.

   name_replacements[ name_index ] = ...
}
```

Kannattaa välillä tulostaa nimille tehdyt korvaajat että voit olla varma ohjelmasi toimivuudesta. Korvaajataulukon voi tulostaa esim. seuraavalla ’foreach’-silmukalla:

```
for ( String name_to_print : name_replacements )
{
   System.out.print( "\n   " + name_to_print ) ;
}
```

Harjoitus 4:

Nyt on enää jäljellä se että taulukossa olevat alaviivalliset nimet pitää ohjelmatiedostossa korvata toisessa taulukossa olevilla korvaavilla nimillä.

Ohjelmassa **Findreplace.java** on olemassa valmis metodi `replace_string_in_file()` jolla voidaan tiedostossa oleva yksittäinen stringi korvata jollakin toisella stringillä. Jos kopioit tämän metodin ja muutat sen seuraavanlaiseksi

```
static void replace_strings_in_file( String  original_file_name,
                                     String[]  strings_to_replace,
                                     String[]  replacement_strings )
{
```

on sinulla käytössä metodi jolla voidaan samalla kertaa muuttaa useampia nimiä jotka on talletettu taulukkoon.

Tässä uudistetussa metodissa tulee muuttaa kohta jossa käsitellään yksittäinen tekstirivi. Tuo yhden tekstirivin käsittely voidaan tehdä seuraavanlaisella silmukalla:

```
for ( int string_index = 0 ;
          string_index < strings_to_replace.length ;
          string_index ++ )
{
   if ( text_line_from_file.contains( strings_to_replace[ string_index ] ) )
   {
      text_line_from_file  =
        text_line_from_file.replace( strings_to_replace[ string_index ],
                                     replacement_strings[ string_index ] ) ;

      System.out.print( "\n \""  +  strings_to_replace[ string_index ]
          +  "\" was replaced with \""  +  replacement_strings[ string_index ]
           +  "\" on line "  +  line_counter  ) ;
   }
}
```

Kun olet tehnyt edellä kuvatun kaltaisen `replace_strings_in_file()` -metodin, sinulla on jo kaikki ’palikat’ koossa ohjelman lopullisen version tekemiseen. Testauksen takia kannattaa ehkä aluksi kommentoida varsinainen tiedostoon talletus pois ohjelmasta. Tiedostoon talletuksen voi aluksi korvata seuraava silmukka joka tulostaa muutetut rivit:

```
      for ( String text_line : modified_text_lines )
      {
         System.out.print( "\n" + text_line ) ;
      }
```

Loppuhuomautus:

Vaikka saatkin ohjelman toimimaan edellä kuvatulla tavalla, on sen toiminnassa mahdollisuus moniin virheisiin. Sillä ei kannata oikeasti yrittää muuntaa ainakaan suuria ohjelmia ennenkuin alkuperäisistä on varmuuskopiot olemassa.

Ohjelman säännöllinen lauseke joka tunnistaa alaviivallisia nimiä pitäisi muuntaa sellaiseksi että se ei muuta nimiä jotka ovat kokonaan isoilla kirjaimilla kirjoitettu. Yleensähän tällaisissa vakioiden nimissä käytetään alaviivaa vaikka muuten nimet olisivatkin camel case -nimiä.

HARJOITUKSIA LocalDate-LUOKALLA

Java 8 -versiosta alkaen Javaan kuuluu `LocalDate`-standardiluokka jonka avulla voidaan tehdä erilaisia päivämääriin liittyviä laskutoimituksia. Tätä kyseistä luokkaa käyttävät esimerkkiohjelmat **Apollo11.java**, **ImportantBirthdays.java**, **BadLuckDays.java**, **Weddingdates.java** sekä **TitanicTimes.java**.

Harjoitus 1:

Tee ohjelma, joka laskee `LocalDate`-luokan avulla nykyisen ikäsi vuosissa, kuukausissa ja päivissä. Tämän voit tehdä kun määrittelet ohjelmaasi `LocalDate`-olioita seuraavaan tapaan

```
LocalDate  my_birthday  =  LocalDate.of( 1977, 07, 14 ) ;
LocalDate  date_now  =  LocalDate.now() ;
```

Ohjelmasta **TitanicTimes.java** näet kuinka kahden `LocalDate`-olion ajallinen etäisyys voidaan laskea. Ohjelmasta **Apollo11.java** näet kuinka otetaan selville minä viikonpäivänä olet syntynyt. Voit ottaa näiden harjoitusten pohjaksi **TitanicTimes.java**-ohjelman. Muuta kuitenkin ohjelmassa käytetyt nimet sellaisiksi että ne ovat kuten esimerkiksi yllä.

Harjoitus 2:

Kopioi sopivasti koodia ohjelmasta **ImportantBirthdays.java** siten että nyt tekeillä oleva ohjelma tulostaa luettelon jossa kerrotaan milloin on tärkeimmät syntymäpäiväsi ja mille viikonpäiville ne sattuvat. Joudut luonnollisesti muuttamaan nimiä siten että saat **ImportantBirthdays.java**-ohjelmassa olevan silmukan toimimaan omassa ohjelmassasi.

Harjoitus 3:

Lisää ohjelmaan ominaisuus, että se tulostaa päivämäärät jolloin olet 10000 ja 20000 päivää vanha. Ihmisen ikä on 10000 päivää kun hänen ikänsä on suurinpiirtein 27 vuotta ja 4 1/2 kuukautta. Tämän ominaisuuden avulla saat sitten uusia juhlimispäiviä normaalien syntymäpäivien lisäksi. Tämän ominaisuuden saat aikaiseksi esim. kun kasvatat silmukassa päivälaskuria ja samalla kasvatat päivällä `LocalDate`-olion sisältöä seuraavaan tapaan

```
//  Aluksi tehdään kopio alkuperäisestä syntymäpäiväoliosta.

LocalDate date_to_increment = LocalDate.of( my_birthday.getYear(),
                                            my_birthday.getMonth(),
                                            my_birthday.getDayOfMonth() ) ;
int  day_counter  =  0 ;

while ( /* sopiva ehto tänne */ )
{
   date_to_increment = date_to_increment.plusDays( 1 ) ;
   day_counter  ++  ;

   if ( ( day_counter  %  10000 )  ==  0 )
   { // sopivaa jatkoa tästä alkaen.
```

Harjoitus 4:

Paranna edellisessä tehtävässä tekemääsi ominaisuutta siten että ohjelma ilmoittaa tarkan ikäsi—vuosina, kuukausina ja päivinä—niinä päivinä jolloin olet 10000 tai 20000 päivää vanha. Tehtyäsi tämän harjoituksen ohjelman tuottama tulostus pitäisi näyttää suurinpiirtein seuraavanlaiselta:

```
10000 days old on 2004-11-29 (Monday)  27 years, 4 months, and 15 days.
20000 days old on 2032-04-16 (Friday)  54 years, 9 months, and 2 days.
```

Harjoitus 5:

Lisää ohjelmaan ominaisuus että se ilmoittaa milloin olet miljardi sekuntia, siis 1000000000 s, vanha. Myös tämän saat tehtyä siten että lasket silmukassa päiviä syntymäpäivästäsi lähtien. Vuorokaudessa on 24 * 60 * 60 sekuntia. Miljardi sekuntia saavutetaan joskun 31 ikävuoden jälkeen. Ohjelmasi pitää lisäksi ilmoittaa tarkka ikäsi vuosina, kuukausina, ja päivinä silloin kun olet miljardi sekuntia vanha.

Tässä harjoituksessa ei tarvitse huomioida tarkkaa syntymähetkeä syntymäpäivänäsi. Voit siis olettaa että jo ensimmäisenä päivänäsi tuli 24 * 60 * 60 sekuntia elettyä. Tämmöinen tarkempi laskenta olisi mahdollista jos käytetään luokkaa `LocalDateTime` jossa voidaan ottaa kellonaika mukaan laskentaan. Jos tarkempaan laskentaan mennään, pitää henkilön sitten tietää myös syntymänsä (tarkka) kellonaika.

HARJOITUKSIA OHJELMALLA SoccerWorldCups.java

SoccerWorldCups.java on esimerkkiohjelma jossa on `ArrayList`-luokkaan perustuvaan taulukkoon talletettu joukko olioita. Tätä ’tietomassaa’ sitten ohjelmassa käsitellään moderneilla Javan tekniikoilla. Ohjelmassa on käytössä mm. Lambda expression.

Harjoitus 1:

Lisää ohjelmaan tiedot tulevista tai taulukosta puuttuvista jalkapallon maailmanmestaruus- eli World Cup -kisoista. Tuleviin kisoihin voi voittajamaaksi laittaa "not known".

Harjoitus 2:

Nykyisellään ohjelma ei sano mitään jos sille annetaan vuosi jona ei jalkapallon World Cupia järjestetty. Muuta ohjelma sellaiseksi että se tulostaa tässä tapauksessa jotain seuraavan kaltaista:

```
"No Wold Cup was organized in ..."
```

Harjoitus 3:

Lisää ohjelmaan menusta valittava toiminto

```
"Find World Cups hosted by a certain country"
```

Harjoitus 4:

Lisää ohjelmaan menusta valittava toiminto jolla etsitään sellaiset `WorldCup`-oliot joissa isäntämaa ja voittajamaa ovat samoja. Tällä toiminnolla voi siis etsiä kisat joissa on saattanut olla kotikenttäetua.

Harjoitus 5:

Käytä `String`-luokan `toLowerCase()`-metodia sopivasti siten että ohjelma osaa etsiä oikean maan nimen vaikka sille annettaisiin maan nimi pienellä alkukirjaimella.

SÄIKEET -- HERÄTYSKELLON RAKENTAMISHARJOITUKSET

Harjoitus 1.

Ohjelmasta Showtime.java näet kuinka Java-ohjelma saa selville nykyisen koneen kellonajan. Tutki tuota ohjelmaa ja tee herätyskello-ohjelma, joka kysyy ensin herätystunnin ja herätysminuutin tyyliin

```
System.out.print( "\n Anna heratystunti:    " ) ;

int  heratystunti     =  Integer.parseInt( keyboard.nextLine() ) ;

System.out.print( "\n Anna heratysminuutti: " ) ;

int  heratysminuutti  =  Integer.parseInt( keyboard.nextLine() ) ;
```

Tämän jälkeen ohjelman tulee silmukassa odottaa että kello tulee herätysaikaan, ja silmukan jälkeen voidaan antaa herätys. Herätys voi olla yksi piippaus. Silmukan sisällä luodaan kalenteriolio jonka avulla saadaan nykyinen aika selville. Silmukan rakenne voi olla seuraavan tapainen

```
int  nykyinen_tunti     =  99 ;
int  nykyinen_minuutti  =  99 ;

while ( nykyinen_tunti    !=  heratystunti  ||
        nykyinen_minuutti !=  heratysminuutti )
{
   GregorianCalendar  aika_juuri_nyt  =  new  GregorianCalendar() ;

   nykyinen_tunti     =  aika_juuri_nyt.get( Calendar.HOUR_OF_DAY ) ;
   nykyinen_minuutti  =   .... // keksi tänne jotain

   // Kannattaa pitää sekunti taukoa ettei prosessori kuumene.

   try
   {
      Thread.sleep( 1000 ) ;
   }
   catch ( InterruptedException vangittu_poikkeus )
   {
   }
}
```

Piippaus saadaan aikaiseksi tulostamalla ns. BEL-merkki "\007"

Harjoitus 2:

Pyydä opettajalta valmis ohjelma Piippaussaie.java käyttöösi. Luo harjoituksessa 1 tekemäsi herätyskello-ohjelman lopussa olio luokasta Piippaussaie ja pistä piippausolio toimimaan kutsumalla metodia start(). Piippausolion toiminnan voi lopettaa kutsumalla metodia stop(). Tarkoitus on että herätyskello-ohjelman lopussa vaadit käyttäjältä Enter-nappulan painamisen ennenkuin lopetat piippausolion tooiminnan.

Säikeiden käytöstä ja luonnista näet esimerkkejä ohjelmassa DotsAndDollars.java.

Enterin painalluksen voi lukea esim. seuraavalla käskylauseella

```
String  turha_stringi  =  keyboard.nextLine() ;
```

Piippaussaie -olion voi luoda antamatta konstruktorille argumentteja. Tällöin olio käyttää oletusarvoista herätysmelodiaa. Olion voi luoda myös siten että antaa konstruktorille argumenttina käytettävän herätysmelodian. "Melodialla" tarkoitetaan tässä piippausten välissä olevien taukojen pituutta. Tutki melodian soiton mekanismia ja "sävellä" kellollesi oma herätysmelodia.

Harjoitus 3:

Edellisen harjoituksen herätyskellon ongelma on että kelloa ei voi pysäyttää ennenkuin se ehtii herätysaikaan. Ongelman voi ratkaista sijoittamalla herätyskellotoiminnan osittain omaan säikeeseensä.

Pyydä opettajalta käyttöön valmis ohjelma Heratyskellosaie.java ja pistä oma herätyskello-ohjelmasi hyödyntämään sitä. (Huomaa että ohjelmassa Herätyskellosaie.java on konstruktori.) Tarkoitus on että "pääohjelmassa" luodaan sekä Piippaussaie- että Heratyskellosaie-oliot sekä tämän jälkeen käynnistetään jälkimmäinen olio. Tämän jälkeen voidaan odottaa käyttäjän Enter-nappulan painallusta riippumatta siitä onko Herätyskellosaie-olio käynnistänyt piippauksen. Lopuksi ohjelmassa pysäytetään molemmat säikeet.

Harjoituksessa 1 alussa tehty herätyskello-ohjelma joudutaan tässä harjoituksessa pistämään suurelta osin uusiksi.