C#-harjoituksia tietokoneluokkaan

Kari Laitinen
http://www.naturalprogramming.com

2005-01-13 Tiedosto luotu.
2007-01-24 Viimeisin muutos

- Älä pidä tiedostoja verkkolevyllä kun teet ohjelmointiharjoituksia.
- Varo antamasta C#-ohjelmallesi nimeä String.cs, koska String on C#-kielen standardiluokka. Tiedosto String.cs työkansiossa voi sotkea muiden ko. kansiossa olevien ohjelmien toiminnan.
- Opiskelumateriaalissa esitellyt suomenkieliset C#-lähdeohjelmat löytyvät hakemiston http://www.naturalprogramming.com/csohjelmat/ alla olevista alihakemistoista.

C#-harjoitus: SILMUKKA JOKA TULOSTAA KERTOTAULUN

1. Tee ohjelma "Kerro.cs" 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 C#ssa * kuten C++:ssakin.

2. Saatuasi aikaan 4:n kertotaulun tulostuksen, paranna ohjelmaa siten että se kysyy käyttäjältä, minkä luvun kertotaulu tulostetaan.

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

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

Tarkoitus on että Main()-metodin sisällä tätä metodia voidaan kutsua esim. seuraavilla tavoilla

```
tulosta_kertotaulu_yms( 4 ) ;
tulosta_kertotaulu( kerrottava_kayttajalta ) ;
```

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. Ohjelmassa Formatointeja.cs on kerrottu kuinka int-arvoja voidaan tulostaa heksadesimaalisena.

LUKUJEN SUMMAUSSILMUKKA

Ota pohjaksi "Sum.cs" ja muuta ohjelma sellaiseksi että se lukee näppäimistöltä kokonaislukuja while- tai do-while-silmukassa, laskee silmukassa lukujen summaa ja näyttää sitä käyttäjälle. Ohjelma lopettaa lukujen kyselemisen kun sille annetaan luku 0 (nolla). Ohjelmassa riittää muuttujat

```
int lukujen_summa     =  0 ;
int luku_nappikselta  =  1 ;
```

Kun luku_nappikselta on alustettu arvolla 1, ohjelmassa voi käyttää while-silmukkaa tyyliin

```
while ( luku_nappikselta  !=  0 )
{
   ...
```

STRINGIEN KÄSITTELYN HARJOITUS

1. C#:ssa on oma luokkansa nimeltä string merkkijonojen talletukseen ja käsittelyyn. Tee ohjelma joka pyytää näppäimistöltä stringin ja, luettuaan stringin, tulostaa sen ensin sellaisenaan ja sitten ’harvasti’ ja ’takaperin harvasti’. Jos ohjelmalle annetaan esimerkiksi stringi "Halonen", sen pitäisi tulostaa

```
Halonen
H a l o n e n
n e n o l a H
```

Stringin saa luettua Console -luokan metodilla ReadLine() esimerkiksi seuraavasti:

```
string annettu_stringi  =  Console.ReadLine() ;
```

Katso mallia ohjelmista Kokonimi.cs, Harvanimi.cs ja Takaperin.cs

2. Lisää ohjelmaan ominaisuus että annettu stringi tulostetaan myös merkkien heksadesimaalisina koodeina. Jos ohjelmalle annettiin stringi "Hello!", sen pitäisi tulostaa

```
48 65 6C 6C 6F 21
```

Tässä tulostuksessa 48 olisi kirjaimen H heksadesimaalinen koodi, 65 kirjaimen e heksakoodi, jne. Stringin yksittäiseen merkkiin voidaan viitata indeksoimalla stringiä. Jos joku_merkki on char-tyypin muuttuja, se voidaan muuttaa heksadesimaaliseksi stringiksi seuraavanlaisella lausekkeella

```
( (int) joku_merkki ).ToString( "X" )
```

3. Lisää ohjelmaan ominaisuus, että se ilmoittaa monesko merkki annetussa stringissä on pieni a-kirjain. Esimerkiksi stringissä "Halonen" a on toinen kirjain. Tässä voi hyödyntää IndexOf() -metodia seuraavaan tapaan

```
int  a_kirjaimen_indeksi  =
                    annettu_stringi.IndexOf( 'a' ) ;
```

Metodi IndexOf() palauttaa arvon -1 jos tutkittavassa stringissä ei ole etsittyä kirjainta tai merkkiä. Ohjelman tulee ilmoittaa mikäli annetussa stringissä ei ole a-kirjainta. Tutki String-luokan dokumentaatiota, ja kokeile IndexOfAny()-metodia jolla voidaan etsiä sellainen paikka stringistä jossa esiintyy jokin annetuista merkeistä.

4. Lisää ohjelmaan ominaisuus että alussa annettu stringi tulostetaan isoilla kirjaimilla. Tämä voidaan tehdä joko muuttamalla kirjaimet yksitellen isoiksi kirjaimiksi siten että kirjainten merkkikoodeista vähennetää 20H jos ne ovat pieniä kirjaimia. Helpommalla tämä homma saadaan aikaiseksi jos käytetään valmista string-metodia nimeltä ToUpper().

5. Tee ohjelman loppuun sananarvauspeli. Ohjelmalle annetaan ensin arvattava sana (tämän sanan voi antaa esim. vieressä istuva henkilö). Tämän jälkeen pelaaja antaa ohjelmalle kirjaimen tai sanan. Jos ohjelmalle on annettu arvattavaksi sanaksi "makkara", pelin kulku voi aiheuttaa esim. seuraavanlaisen tulostuksen:

```
_______ Anna kirjain tai sana : a

_a__a_a Anna kirjain tai sana : m

ma__a_a Anna kirjain tai sana : r

ma__ara Anna kirjain tai sana :
```

Tässä siis näytetään alaviivalla ne sanan kirjaimet joita ei ole vielä arvattu. Tässä kannattaa käyttää StringBuilder-stringiä arvaustilanteen talletukseen koska StringBuilder-olion sisältämät stringit voivat muuttua ohjelman suorituksen aikana. StringBuilder-stringiä on käytetty esim. ohjelmassa Juice.cs. Ohjelman tulee myös ilmoittaa jos käyttäjä antoi arvattavaan sanaan kuulumattoman kirjaimen. Jos käyttäjä antoi sanan, tulee ilmoittaa onko se oikein vai väärin. Sananarvauspelin ohjelmakoodin alkua on annettu jonkin verran seuraavalla sivulla:

```
//  Sananarvauspelin alku

Console.Write( "\n\n SANANARVAUS. Anna arvattava sana: " ) ;

string  arvattava_sana  =  Console.ReadLine() ;

Console.Write( "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n" ) ;

StringBuilder  arvaustilanne  =  new StringBuilder(
                    new String( '_', arvattava_sana.Length ) ) ;

string  arvaus = "" ;

bool  peli_on_pelattu  =  false ;

while ( peli_on_pelattu  ==  false )
{
   Console.Write( "\n " +  arvaustilanne  +  " Anna kirjain tai sana : "  ) ;

   arvaus  =  Console.ReadLine() ;

   if ( arvaus.Length  ==  1 )
   {
```

HARJOITUKSIA OHJELMALLA Takaperin.cs

Harjoitus 1:

Lisää ohjelman loppuun silmukka joka laskee ja tulostaa taulukossa olevien lukujen summan. Ohjelmassa on alunperin kaksi while-silmukkaa, mutta voit poistaa jälkimmäisen silmukan, ja pistää sen tilalle uuden silmukan. Ohjelma voi jatkua ensimmäisen while-silmukan jälkeen seuraavasti

```
int lukuja_annettiin  =  luvun_indeksi ;
int lukujen_summa     =  0 ;

for ( luvun_indeksi  =  0 ;
      luvun_indeksi  <  lukuja_annettiin ;
      luvun_indeksi  ++ )
{
   lukujen_summa  =  lukujen_summa + ...

}
```

Harjoitus 2:

Lisää ohjelmaan ominaisuus, että lasketun summan perusteella lasketaan ja tulostetaan annettujen lukujen keskiarvo. Tarkan keskiarvon laskemiseksi voi int-muuttujien arvot muuttaa tilapäisesti liukuluvuiksi. Tämä voidaan tehdä kirjoittamalla (float) muuttujan nimen eteen.

Harjoitus 3:

Tee summan laskemista varten seuraavanlainen metodi

```
public static int laske_summa( int[] annettu_taulukko,
                               int   lukujen_maara )
{
   int  laskettu_summa  =  0 ;

   ...

   return laskettu_summa ;
}
```

Em metodia on tarkoitus kutsua main() -metodin sisältä

Harjoitus 4:

Tee myös keskiarvon laskemisesta oma metodi seuraavaan tapaan

```
public static float laske_keskiarvo(
                           int[] annettu_taulukko,
                           int   lukujen_maara )
{
   float  laskettu_keskiarvo  =  0 ;

   ...
```

HARJOITUKSIA OHJELMALLA Elaimia.cs

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-propertyä Length tarkistamaan onko vatsa tyhjä. Propertyä Length 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".

Tämänkin tehtävän onnistumisen voit todeta anna_puhua() -metodin avulla.

Harjoitus 5:

Lisää luokkaan toinen versio metodista ruoki() siten että tällä uudella metodilla voi syöttää toisen eläimen jollekin Elain-oliolle. Tämä uusi ruoki()-metodi voi alkaa seuraavasti:

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

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

```
syotava_elain.elaimen_nimi
```

Syödyn eläimen datakenttä elaimen_nimi voi muuttua syömisoperaatiossa nimelle "Syoty elain". Kun uusi ruoki()-metodi on olemassa pitäisi lauseiden

```
Elain  tiikeri  =  new Elain( "tiikeri", "Richard" ) ;
Elain  nauta    =  new Elain( "nauta", "Mansikki" ) ;

tiikeri.ruoki( nauta ) ;
tiikeri.anna_puhua() ;
```

tuottaa tulostus

```
Hei. Mina olen tiikeri nimelta Richard.
Olen syonyt: Mansikki
```

Harjoitus 6:

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
```

HARJOITUKSIA Paivamaara-LUOKALLA

Harjoitus 1:

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

```
Paivamaara syntymaaika = new Paivamaara( "29.11.1981" ) ;
Paivamaara paivamaara_nyt  =
                      new  Paivamaara( "04.03.2005" ) ;
```

Ohjelmasta Talvisota.cs näet kuinka kahden Paivamaara-olion ajallinen etäisyys voidaan laskea. Voit myös helposti ohjelman avulla ottaa selville minä viikonpäivänä olet syntynyt. Jos otat pohjaksi Talvisota.cs-ohjelman, muuta ohjelmassa käytetyt nimet sellaisiksi että ne ovat kuten esimerkiksi yllä.

Harjoitus 2:

Lisää Paivamaara-luokkaan kopiointikonstruktori, jolla Paivamaara-olio voidaan kopioida esim. seuraavasti

```
Paivamaara kasvatettava_paivays  =
                    new  Paivamaara( syntymaaika ) ;
```

Kopiointikonstruktori on esim. aiemmin nähdyssä Elain-luokassa ohjelmassa Elaimia.cs. Paivamaara-luokan tapauksessa kopiointikonstruktorin tulee kopioida neljä datakenttää.

Harjoitus 3:

Kun kopiointikonstuktori on olemassa, lisää kohdassa 1 tekemääsi ohjelmaan ominaisuus jolla voit laskea ikäsi syntymäpäivästä nykyiseen päivään. Tämä voidaan tehdä jos syntymäajan sisältävä Paivamaara-olio ensinnä kopioidaan kopiointikonstruktorilla ja sen jälkeen lasketaan päiviä seuraavaan tapaan

```
int   paivalaskuri  =  0 ;

while  (  kasvatettava_paivays.on_aikaisempi_kuin(
                                          paivamaara_nyt ) )
{
   kasvatettava_paivays.kasvata() ;
   paivalaskuri  ++ ;
}

Console.Write( "\n Olet nyt "
            +  paivalaskuri + " paivaa vanha.\n" ) ;
```

Harjoitus 4:

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 kun lasket silmukassa päiviä samaan tapaan kuin edellisessä kohdassa tehtiin.

Harjoitus 5:

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

Harjoitus 6:

Paranna ohjelmaasi vielä siten että otat käyttöön valmiin PaivamaaraNyt-luokan ja määrittelet olion johon paivamaara_nyt viittaa tuon luokan avulla. Kun teet näin, ohjelmasi pitäisi toimia siten että aina kun sen suoritat tietokoneessa, se automaattisesti ilmoittaa senhetkisen tarkan ikäsi. (Tietokoneen kellon ja kalenterin pitää luonnollisesti olla oikeassa ajassa että tämä olisi mahdollista.)

HARJOITUKSIA OHJELMALLA Presidentit.cs

Kopioi ohjelmatiedostot Presidentit.cs ja Paivamaara.cs paikalliseen työkansioosi ja testaa ensin ohjelman toimintaa.

Harjoitus 1:

Lisää ohjelmaan Suomen seuraavan presidentin tiedot, ja testaa että ohjelma osaa käyttää lisäämiäsi tietoja. (Jos et jostain syystä tiedä kenestä tulee Suomen seuraava presidentti, voit luonnollisesti kuvitella siihen virkaan itsesi.)

Harjoitus 2:

Paranna Presidentti-luokan metodia lue_kaikki_presidentin_tiedot() siten että sen palauttamassa stringissä annetaan tieto siitä minkä ikäinen kyseisen Presidentti-olion edustama presidentti oli astuessaan virkaansa. Voit käyttää tuon iän laskentaan Paivamaara-luokan metodia laske_etaisyys(). Lisäksi tarvitset seuraavanlaisia muuttujia

```
int ian_vuodet, ian_kuukaudet, ian_paivat ;
```

Harjoitus 3:

Presidenttitietosovellus-luokassa on metodi nimeltä tulosta_seuraavan_presidentin_tiedot() jonka avulla presidenttien tietoja voidaan "kelata" eteenpäin. Tee kyseiseen luokkaan vastaava metodi tulosta_edellisen_presidentin_tiedot() jonka avulla presidenttien tietoja voidaan "kelata" taaksepäin. Tätä uutta metodia pitäisi sitten kutsua suorita()-metodista.

Harjoitus 4:

Paranna suorita()-metodin while-silmukkaa siten että ohjelma ei kaadu siinä tapauksessa jos käyttäjä painaa pelkkää Enter-näppäintä silloin kun sitä pyydetään valitsemaan toiminto.

Harjoitus 5:

Johda (periytä) Presidentti-luokasta uusi luokka nimeltä PresidenttiEnglish ja tee tähän luokkaan uudet versiot metodeista lue_lyhyt_presidenttiinfo() ja lue_kaikki_presidentin_tiedot() ja tee näistä metodeista sellaisia että ne sijoittavat kaikki tietoihin lisättävät tekstit englanniksi. Nämä metodit tulee määritellä Presidetti-luokassa varatulla sanalla virtual jotta niiden ylikirjoittaminen on mahdollista. Uusi luokka tulee olemaan seuraavantapainen

```
class PresidenttiEnglish : Presidentti
{
   //  konstruktori joka kutsuu yläluokan
   //  konstruktoria

   //  versio metodista lue_lyhyt_presidenttiinfo()

   //  versio metodista lue_kaikki_presidetin_tiedot()
}
```

Esimerkiksi ohjelmasta PankkiMonimuotoinen.cs näet kuinka ylemmän luokan metodeita uudelleenmääritellään alemmassa luokassa.

Uuden PresidenttiEnglish-luokan olioita pitäisi voida käyttää presidenttitaulukossa tekemättä muutoksia muuhun ohjelmaan. Esimerkiksi jos ohjelmaan muutetaan

```
presidenttitaulukko[ 0 ]  =  new
 PresidenttiEnglish( "Kaarlo Juho Stahlberg",
                     "28.01.1865",  "Suomussalmi",
                     "Keskusta-oikeisto",
                     "26.07.1919", "01.03.1925" ) ;
```

K.J. Ståhlbergin tiedot tulostuvat automaattisesti englanniksi.

HARJOITUKSIA LUOKKIEN PERINTÄÄN LIITTYEN

Harjoitus 1:

Lisää ohjelmaan PankkiMonimuotoinen.cs uusi luokka nimeltä RikkaanTili, joka on sellainen että siltä ei voi nostaa kuin tietyn summan ylittäviä rahamääriä. Tarkoitus on että johdat (periytät) RikkaanTili-luokan luokasta Pankkitili ja kirjoitat siihen uuden version metodista talleta_rahaa(). Tuo metodi on määritelty jo varatulla sanalla virtual Pankkitili-luokassa joten sen voi uudelleenkirjoittaa alemmassa luokassa. RikkaanTili-luokassa tulee olla datajäsen

```
protected double talletusraja ;
```

ja sellainen konstruktori joka asettaa talletusrajan. Jos esimerkiksi kirjoitetaan

```
RikkaanTili  sikarirahatili  =
             new  RikkaanTili( "Peikko Pesonen", 998877,
                               400000.00,  10000.00 ) ;
```

määritellään tiliolio jonka alkupääoma on 400000.00 ja jolle tehtävien talletuksien tulee olla vähintään 10000.00.

Harjoitus 2:

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.

Harjoitus 1:

Tee tekstieditoriohjelma (esim. Editor.cs) joka lukee näppäimistöltä annettuja tekstirivejä Console.ReadLine()-metodin avulla ja tallettaa annetut tekstirivit tekstitiedostoon. 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.

Tee tekstieditoriohjelmasta aluksi sellainen että se pistää annetut tekstirivit aina samannimiseen tekstitiedostoon (esim. tekstia.txt)

Editori kannattaa rakentaa siten että käyttäjän antamat tekstirivit pannaan aluksi talteen ArrayList-taulukkoon, ja sitten kun käyttäjä on lopettanut tekstin antamisen, eli kun saatiin rivi jonka ensimmäinen merkki on piste, ArrayList-taulukossa olevat tekstirivit pannaan tiedostoon taltteen. Kun katsot mallia ohjelmasta Etsikorvaa.cs, näet kuinka ArrayList-luokkaan perustuvaa taulukkoa käytetään tekstirivien talletukseen. Itse asiassa tuossa ohjelmassa on valmis metodi jolla saat sitten tekstin talletettua ArrayList-taulukosta tiedostoon. ArrayList-luokkaan perustuva taulukko luodaan esim. seuraavasti

```
ArrayList  annetut_tekstirivit  =  new  ArrayList() ;
```

Silmukka, joka lukee tekstirivejä näppäimistöltä voi alkaa seuraavasti

```
bool  tekstia_halutaan_viela_antaa  =  true ;
while ( tekstia_halutaan_viela_antaa  ==  true )
{
   string tekstirivi_nappaimistolta = Console.Readline();
   ...
```

Silmukan sisällä tulee testata, onko tekstirivin ensimmäinen merkki piste ja jos on niin bool-muuttuja asetetaan arvoon false, jolloin silmukan suoritus loppuu.

Harjoitus 2:

Paranna editoria siten että käytettävä tekstitiedoston nimi voidaan antaa komentoriviparametrina tai jos sitä ei annettu komentoriviltä, niin ennen tekstin talletusta tiedostoon se kysytään käyttäjältä. Ohjelmasta Etsi.cs näet kuinka komentoriviparametreja voidaan käyttää.

Harjoitus 3:

Paranna edellisessä osatehtävässä tekemääsi ominaisuuttaa siten että ennen tiedostoon kirjoitusta testaat onko jo annetun niminen tiedosto olemassa. Jos se on jo olemassa, käyttäjältä kysytään saapiko olemassaolevan tiedoston päälle kirjoittaa, ja jos käyttäjä vastaa kieltävästi, pitää käyttäjältä kysyä uusi tiedostonimi. Tiedoston olemassaolo pitäisi voida testata File.Exists()-metodin avulla. Tässä saattaisi olla tarpeen käyttää silmukkaa

```
bool  hyvaksyttava_tiedoston_nimi_saatu  =  false ;

while ( hyvaksyttava_tiedoston_nimi_saatu  ==  false )
{
   ...

   if ( File.Exists( annettu_tiedoston_nimi ) )
   {
      ...
   }
   else
   {
      ...
   }

   ...
}
```

Harjoitus 4:

Paranna vielä ohjelmaa siten että jos käyttäjä antoi tiedostonimen jonka niminen tiedosto on jo olemassa, niin käyttäjälle tarjotaan mahdollisuus kirjoittaa antamansa tekstirivit olemassaolevan tekstitiedoston perään. Tekstitiedoston perään voidaan liittää tekstiä kun tiedosto avataan seuraavasti

```
StreamWriter  kirjoitettava_tiedosto  = new
       StreamWriter( annettu_tiedoston_nimi, true ) ;
```

HARJOITUKSIA OHJELMAAN Kaanna.cs LIITTYEN

Kaanna.cs on ohjelma joka osaa kääntää luonnollisen kielen sanoja sen mukaan mitä käännösolioita on talletettu ArrayList-taulukkoon kaannostaulukko.

Harjoitus 1:

Lisää ohjelmaan ominaisuus että se ilmoittaa käyttäjälleen jos se ei kykene suorittamaan sille annettua käännöstehtävää. Metodi kaanna() palautaa true- tai false-arvon sen mukaan onnistuiko annetun sanan käännös, mutta nykyisessä ohjelman versiossa tuota tietoa ei käytetä hyödyksi. (Metodia kaanna() kutsutaan Main()-metodista aivan kuin se olisi void-tyyppinen metodi, vaikka tosiasiassa se on bool-tyyppinen metodi.) Jotta tässä osatehtävässä vaaditun ominaisuuden saa tehtyä, voi Main()-metodiin määritellä muuttujan

```
bool sana_on_saatu_kaannettya  =  false ;
```

joka merkataan arvoon true sitten kun jokin kaanna()-metodin kutsu onnistuu käännöstyössään.

Harjoitus 2:

Lisää ohjelmaan ominaisuus että se tulostaa kaikki käännöstaulukossa olevat käännöstiedot ruudulle siinä tapauksessa että käyttäjä ei anna komentoriviltä sanaa. Tätä varten sekä KaksikielinenKaannos- että KolmikielinenKaannos-luokkaan on tehtävä metodi tulosta() jolla käännös voidaan tulostaa ruudulle. Tällaiset metodit saa kätevästi aikaiseksi kun kopioi luokista kaanna()-metodit ja yksinkertaistaa niitä. (Voit tietysti tehdä tulosta()-metodin sijaan luokkiin ToString()-nimiset metodit, jolloin olioiden tulostaminen on vieläkin helpompaa.)

Harjoitus 3:

Lisää ohjelmaan ominaisuus että kaannostaulukko-taulukossa olevat käännösoliot pannaan aakkosjärjestykseen datakentän ensimmainen_sana suhteen. Tätä varten on katsottava mallia ohjelmasta Tapahtumia.cs. KaksikielinenKaannos-luokkaan on tehtävä CompareTo()-metodi aivan samaan tapaan kuin Tapahtumia.cs-ohjelman Tapahtuma-luokassa on tehty. KaksikielinenKaannos-luokan tulee siis toteutaa IComparable-rajapinta. Kun tuo CompareTo()-metodi on olemassa, voidaan ArrayList-luokkaan perustuvan taulukon oliot pistää aakkosjäjestykseen ArrayList-luokan Sort()-metodia kutsumalla.

Toteutettavan CompareTo()-metodin tulee vertailla "tämän" olion ensimmainen_sana-kentää parametrina annetun olion vastaavaan kenttään. Koska ensimmainen_sana-kenttä on stringi, voit käyttää String-luokan CompareTo()-metodia vertailun suorittamiseen. String-luokan CompareTo()-metodi palauttaa automaattisesti "oikeat" palautusarvot mukaan miten vertailtavat stringit suhtautuvat toisiinsa. Noiden palautusarvojen avulla stringit pitäisi mennä automaattisesti aakkosjärjestykseen silloin kun Sort()-metodia käytetään.

HARJOITUKSIA OHJELMALLA PisteitaJaProsentteja.cs

Harjoitus 1:

Muuta main()-metodi sellaiseksi että säikeiden lopettamiseksi vaaditaan kaksi Enterin painallusta. Ensimmäisen Enterin painalluksen jälkeen lopetetaan pisteitä tulostava säie. Toinen Enterin painallsu lopettaa prosentteja tulostavan säikeen ja myös itse main()-metodia suorittavan säikeen.

Harjoitus 2:

Muuta pisteitä tulostava säie sellaiseksi että jokaisen pisteen tulostuksen jälkeen pisteiden välillä pidettävä tauko pienenee 50 millisekuntia. Voit määritellä luokkaan datajäsenen

```
long tauko_pisteiden_valilla  =  1000 ;
```

ja pienentää tämän arvoa jokaisen tulostuksen jälkeen.

Koska tässä on vaarana että tauko menee negatiiviseksi, pitää ohjelmaan laittaa ominaisuus että tauko palaa arvoon 1000 sen jälkeen kun se on mennyt pienemmäksi kuin 100 millisekuntia. Tässä voidaan käyttää if-rakennetta

```
if ( tauko_pisteiden_valilla  <  100 )
{
   tauko_pisteiden_valilla  =  1000 ;
}
```

Harjoitus 3:

Ohjelmasta Countdown.cs näet että ylöspäin millisekunteja laskeva systeemikello voidaan lukea metodin System.currentTimeMillis() avaulla. Käytä tätä metodia ja testaa kuinka kauan prosentteja tulostavassa säikeessä 4.05 sekunnin tauko todellisuudessa kestää. Ajan saa laskettua kun pistää käskyn

```
long aika_ennen_taukoa  =  System.currentTimeMillis() ;
```

ennen try-catch -rakennetta ja käskyn

```
long aika_tauon_jalkeen  =  System.currentTimeMillis() ;
```

try-catch -rakenteen jälkeen.

Näiden long-muuttujien erotuksen voi sitten tulostaa prosenttimerkin sijaan.
Jos pistät Countdown.cs -ohjelman toimimaan yhtä aikaa muutetun PisteitaJaProsentteja.cs -ohjelman kanssa, huomaat että laskettu tauon pituus kasvaa.

Seuraavan HARJOITUS 4:n OMINAISUUDET EI TOIMINEET KÄYTÄNNÖSSÄ!

Harjoitus 4:

Lisää ohjelmaan erillinen hidastussäie, jonka pistät toimimaan muiden säikeiden rinnalle. Hidastussaie -luokan pohjaksi voi kopioida esim. SaiePisteidenTulostukseen -luokan. Tarkoitus on että hidastussäie aiheuttaa hitautta suorittamalla run() -metodin while-silmukan sisällä jotain hidastelusilmukkaa. Hidastelusilmukka voi olla seuraavan tapainen

```
long odotettava_aika  =
            System.currentTimeMillis()  +  5000 ;

while ( System.currentTimeMillis() <  odotettava_aika )
{
   ;
}
```

Hidastelusilmukan jälkeen ohjelmassa tulee olla try-catch -rakenne jossa kutsutaan Thread.sleep() -metodia jolla aiheutetaan 5 sekunnin nukkuminen. Hidastussaie siis ensin hidastaa koko konetta 5 sekunnin ajan ja sitten nukkuu toiset viisi sekuntia. Hidastussaikeen vaikutuksen huomaa sitten prosenttisäikeen tulostuksesta.

HERÄTYSKELLON RAKENTAMISHARJOITUKSET

Harjoitus 1.

Tee ohjelman "ClockExperiments.cs" pohjalta herätyskello-ohjelma, joka kysyy ensin herätystunnin ja herätysminuutin tyyliin

```
System.out.print( "\n Anna heratystunti: " ) ;
int waking_hour  =  Keyboard.get_int() ;

System.out.print( "\n Anna heratusminuutti: " ) ;
int waking_minute  =  Keyboard.get_int() ;
```

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  current_hour    =  99 ;
int  current_minute  =  99 ;

while ( current_hour    !=  waking_hour  ||
        current_minute  !=  waking_minute )
{
   Calendar calendar_today  =  new Gregorian....


   current_hour    =  calendar_today.get(
                         Calendar.HOUR_OF_DAY ) ;

   current_minute  =  ...
}

// täällä voidaan sitten antaa herätys
```

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

Harjoitus 2:

Ota valmis ohjelma Piippaussaie.cs 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 ThreadTesting.cs.

Enterin painalluksen voi lukea esim. seuraavalla käskylauseella

```
String turha_stringi  =  Keyboard.get_string() ;
```

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.

Ota käyttöön valmis ohjelma Heratyskellosaie.cs ja pistä oma herätyskello-ohjelmasi hyödyntämään sitä. (Huomaa että ohjelmassa Herätyskellosaie.cs 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.