Syöttö ja tulostus
Tietotyypit
Parametrien välitys

Olio-ohjelmointi

Lapin AMK / EM

Ensimmäinen C#-ohjelma

```
using System;
namespace Hello
{
  public class Lukuvuosi
  {
    private int alku = 2020;
    private int loppu = 2021;

    public static void Main(String[] args)
    {
      Lukuvuosi lv = new Lukuvuosi();
      Console.WriteLine("Hyvää lukuvuotta {0}-{1}", lv.alku,
     lv.loppu);
    }
  }
}
```

Nimiavaruus ja luokka

- **using System**
  - Ohjelmassa käytetään nimiavaruuden System luokkia, tässä Console-luokkaa
- **namespace Hello**
  - Ohjelmakoodi tässä tiedossa kuuluu nimiavaruuteen Hello
  - Nimiavaruus on joukko yhteenkuuluvia luokkia
- **public class Lukuvuosi**
  - Esitellään julkinen luokka Lukuvuosi
  - Luokka on abstrakti tietotyyppi, joka sisältää datan ja sitä käsittelevät operaatiot

Jäsenmuuttuja

- **private int alku = 2020;**
  - Esitellään ja alustetaan kokonaislukutyyppinen muuttuja **vuosi**
  - Kun muuttuja esitellään luokan sisällä, sitä kutsutaan luokan jäsenmuuttujaksi
  - Jäsenmuuttujaan voi viitata suoraan kaikissa saman luokan jäsenfunktioissa eli metodeissa
  - Suojausmääre **private** kertoo, että muuttuja on käytettävissä vain saman luokan sisällä (jäsenfunktioissa)

Tunnisteet

- Luokkien ja muuttujien nimet on tunnisteita (identifier)
  - Tunniste voi sisältää kirjaimia, numeroita sekä alaviivan (_) tai dollarimerkin ($)
  - Tunniste ei voi alkaa numerolla eikä se voi sisältää välilyöntejä
  - Tunnisteessa isot ja pienet kirjaimet merkitsevät (case sensitive)
    - A1 ja a1 ovat siis eri tunnisteita

Main-metodi

- **public static void Main(String[] args)**
  - .NET-ajoympäristö kutsuu Main-metodia, kun ohjelman suoritus aloitetaan
  - Metodin otsikon tulee olla täsmälleen yllä mainitun kaltainen, vain parametrin args nimeä voi muuttaa
- **`void`** kertoo, ettei metodilla ole paluuarvoa
- **`static`** kertoo, että main on staattinen metodi eli luokkametodi
  - Staattista metodia voi kutsua luomatta ensin luokan tyyppistä oliota

Olion luominen

- **Lukuvuosi lv = new Lukuvuosi();**
- Esitellään luokan **Lukuvuosi**-tyyppinen muuttuja
  - **Lukuvuosi lv**
- Luodaan sitten uusi Lukuvuosi-tyyppinen olio
  - **new Lukuvuosi();**
- Lopuksi sijoitetaan viite olioon muuttujan **lv**-arvoksi
  - **lv =…**

Konsolille tulostaminen

- **`Console.WriteLine("Hyvää lukuvuotta {0}-{1}", lv.alku, lv.loppu);`**
- Konsolille voi tulostaa vakiotekstiä ja muuttujien arvoja Console.WriteLine() -metodilla.
- Tulostettava teksti annetaan lainausmerkeissä metodin parametriksi
- Muuttujan tulostuskohta merkitään aaltosuluissa annettavalla muuttujan järjestysnumerolla
  - Numerointi alkaa nollasta
  - Muuttujien nimet luetellaan tekstin jälkeen pilkulla erotettuina

Konsolilta lukeminen: Summa-esimerkki

```
public class Program {
      public static void Main(string[] args) {
         int summa = 0;
         Console.Write("Anna 1. luku:");
         if (int.TryParse(Console.ReadLine(), out int luku1))  {
            summa = summa + luku1;
            Console.Write("Anna 2. luku:");
            if (int.TryParse(Console.ReadLine(), out int luku2)  {
               summa = summa + luku2;
               Console.WriteLine("Lukujen summa on {0}", summa);
            }
            else {
               Console.WriteLine("Anna vain kokonaislukuja!");
            }
         }
         else {
            Console.WriteLine("Anna vain kokonaislukuja!");
         }
      }
   }
}
```

Syötteen lukeminen

- Konsolilta luettava tieto on aina merkkimuotoista (string)
- Syöte voidaan muuttaa halutun tyyppiseksi kyseisen tyypin parse-metodilla
- parse-metodi voi aiheuttaa FormatExceptionin, jos syöttötieto on väärän muotoista
- Tietotyypistä riippuen tyyppikonversiossa voi tapahtua myös ArgumentOutOfRangeException tai OverflowException. Poikkeusten käsittelyyn palataan myöhemmin

Merkkijonon muuntaminen numeeriseksi

- **if (int.TryParse(Console.ReadLine(), out int luku1))**
- Kevyempi tapa merkkijonon muuttamiseen numeeriseksi on käyttää **TryParse**-metodia
- Metodille annetaan parametrina parsittava merkkijono ja muuttuja, johon parsinnan tulos tallennetaan
- Metodin paluuarvo on true tai false sen mukaan, onnistuiko parsinta

Tietotyypit

- Tietotyyppejä on kahdenlaisia: arvotietotyyppejä ja viitetietotyyppejä
- C#:ssa arvotietotyyppejä on kahdenlaisia: enum- ja struct-tyyppejä
- Edelleen C#:ssa struct-tyyppejä on kahdenlaisia:
  - Itsemääriteltyjä struktuureja
  - Ohjelmointikielessä määriteltyjä tyyppejä (simple struct types)

Perustietotyypit

- C#:ssa kaikki perustietotyypitkin ovat struct-tyyppiä (simple struct types)
  - Sen vuoksi perustietotyyppisiin muuttujiin voidaan kohdistaa metodikutsuja, esimerkki:
    ```
    int i = 1;
    string s = i.ToString();
    ```
  - C#-kääntäjä kuitenkin mahdollistaa vakioarvon sijoittamisen perustietotyyppisiin muuttujiin C-tyylisesti, esim. int i = 1;
- C#:n perustietotyypit ovat bool, sbyte, ushort, uint, ulong, byte, short, int, long, decimal, float, double ja char

C#:n perustietotyypit

- Totuusarvotyyppi bool (System.Boolean) voi saada vain arvot true tai false
- Pienin kokonaislukutyyppi on byte ja sen **etumerkillinen** vastine sbyte
  - Nimeäminen poikkeaa muista kokonaislukutyypeistä
- Muita kokonaislukutyyppejä ovat short, int, long ja niiden **etumerkittömät** vastineet ushort, uint, ulong

Perustietotyypit

- CLS ei sisällä sbyte, ushort, uint, ulong tietotyyppejä, joten niitä ei tule käyttää luokan julkisessa rajapinnassa (public ja protected)
- Reaalilukutyyppejä ovat decimal, float ja double.
  - decimal, 128-bittiä, arvoalue ± $7.9 * 10^{28}$, 28 merkitsevää numeroa
  - float, 32-bittiä, arvoalue ± $3.4 * 10^{38}$, 7 merkitsevää numeroa
  - double, 64-bittiä, arvoalue ± $1.7 * 10^{308}$, 15 merkitsevää numeroa

Perustietotyypit

- Ilman tyypintunnusta annettu reaalilukuvakio (esim. 10.0) tulkitaan double-tyyppiseksi
- Vakion perään voidaan liittää type suffix ilmaisemaan vakion tietotyyppi:
  - D/d = double, esim. 5D
  - M/m = decimal, esim. 1.5M
  - F/f = float, esim. 1.5F

Merkkityyppi

- C# käyttää 16-bittistä unicode-merkistöä.
- Yhden kirjainmerkin sisältävä tietotyyppi on char (System.Char)
- System.Char –struktuurin metodeja: IsDigit(), IsLetter(), IsWhiteSpace(), ToLower(), ToUpper(), ToString()

Enum-tyyppi

- Enum-tyyppi eli lueteltu tyyppi on toteutukseltaan kokonaisluku, mutta sen mahdolliset arvot on nimetty/lueteltu
- Enum-muuttujaan voidaan sijoittaa mikä tahansa kokonaislukuarvo, ei pelkästään lueteltuja arvoja
  - Ei ole hyvää ohjelmointityyliä
- Oletuksena Enum-vakioiden arvot alkavat nollasta ja kasvavat siitä eteenpäin yhdellä

Enum-tyyppi

- Esimerkki enum-tietotyypin määrittelystä:

```
enum Maa {
   hertta, ruutu, pata, risti
}
```

- Esimerkki muuttujan määrittelystä ja alustamisesta:

```
Maa m = Maa.ruutu;
```

- Lauseen suorittamisen jälkeen m:n arvo kokonaislukuna on 1

Enum-tyyppi

- Kun enum-tyyppisen muuttujan arvo tulostetaan, tulostetaan sitä vastaava nimi, ei numeroa
- Kaikki enum-tyypin määrittelyssä luetellut arvot voidaan tulostaa seuraavasti (Maa-tyypin määrittely edellisellä kalvolla):

```
string[] apuTaulu;
apuTaulu = Enum.GetNames(typeof(Maa));
foreach (string s in apuTaulu) {
  Console.WriteLine(s);
}
```

Parametrien välitys: arvo- ja viiteparametrit

- Parametreja on kahdenlaisia: arvo- ja muuttujaparametreja
- Arvoparametreista välitetään kutsuttavalle funktiolle kopio eikä funktiossa kopioon tehdyt muutokset välity kutsuvalle funktiolle
- Muuttuja- eli viiteparametrin käyttö merkitsee sitä, että metodi käsittelee suoraan viitteen osoittamaa muistipaikkaa
  - Ts. kutsuttavalle metodille välitetään alkuperäisen muuttujan osoite, ei arvoa
- C#:ssa parametrit ovat oletuksena arvoparametreja

Ref-viiteparametri

- Parametri määritellään viiteparametriksi ref-määreellä, joka pitää olla sekä kutsuttavassa metodissa että itse kutsussa
  - Funktion määrittely: public void laske(ref int luku){…}
  - Funktion kutsu: laske(ref numero);
- Viiteparametrina (ref) ei voi olla vakio (const)
  - Arvoa pitää voida muokata kutsuttavassa metodissa

Out-parametri

- Out-parametri toimii kuten ref-parametri, mutta parametrilla ei tarvitse olla arvoa ennen kutsua, vaan kutsuttu metodi asettaa sille arvon
- Summa esimerkissä käytettiin out-parametria:

if (int.TryParse(Console.ReadLine(), out int luku1))

  - Miksi out-parametri soveltuu tähän tilanteeseen?