Ohjelman rakenne
Suojausmääreet

Olio-ohjelmointi

Lapin AMK / EM

Luokkien rakenne

- Tärkeitä periaatteellisia asioita ovat mm.
  - Jäsenmuuttujat (member variables)
  - Ominaisuudet (properties)
  - Rakentajametodit (constructors)
  - Jäsenfunktiot eli metodit (methods)
  - Suojausmääreet (access specifiers, visibility identifiers)

Opintojakso-luokka

- Esimerkissä määritellään Opintojakso-luokka, joka sisältää kurssin nimen ja laajuuden
- Tässä esitetään aluksi perinteinen toteutus ilman ominaisuuksia, käyttäen erillisiä set/aseta ja get/hae-metodeja

Esimerkki 4-1 Opintojakso-luokka

```
public class Opintojakso {
  private String nimi;
  private int opintopisteet;

  public Opintojakso(String n,int p)
  {
     asetaNimi(n);
     asetaLaajuus(p);
  }

  public void asetaNimi( String n )
  {
     nimi = n;
  }

  public String haeNimi(){
     return nimi;
 }
```

```
  public void asetaLaajuus(int p)
  {
    if (p >= 0) {
       opintopisteet = p;
    }
    else {
       // Jätetään muuttamatta
    }
  }

  public int haeLaajuus() {
     return opintopisteet;
  }
}
```

Rakentaja

- Rakentaja (constructor) on erikoistunut metodi, jota kutsutaan olion luomisen yhteydessä
- Tyypillisesti rakentajametodi alustaa luokan jäsenmuuttujien arvot
- Rakentajan nimi on aina sama kuin sen luokan nimi, johon se kuuluu
- Samalla luokalla voi olla useampia rakentajia, jotka poikkeavat toisistaan joko parametrien määrän, tyypin tai molempien suhteen

Rakentaja

- Jos luokalle ei kirjoiteta lainkaan rakentajaa, kääntäjä luo sille parametrittoman oletusrakentajan
  - Parametritonta oletusrakentajaa ei lisätä, jos luokassa on yksikin muu rakentaja
- Rakentajaa kutsutaan, kun luodaan uusi olio eli luokan ilmentymä. Olio luodaan operaattorilla **`new`**
- Olioita käsitellään aina viitteiden kautta. Olion nimi on viite pysyvällä muistialueella sijaitsevaan olioon

Ominaisuudet (properties): määrittely

- Lyhennysmerkintä julkisten set- ja get-metodien määrittelemiselle private-tyypppisille muuttujille

```
public class Henkilo {
   private int hloIka = 20;
   public int Ika
   {
      get { return hloIka;}
      set { hloIka = value;}
   }
}
```

Ominaisuudet (properties): käyttö

- Luokan määrittelemiin ominaisuuksiin voidaan viitata kuin ne olisivat muuttujanimiä
- Edellisessä esimerkissä esiteltyä ominaisuutta voitaisiin käsitellä näin:

```
Henkilo h = new Henkilo();
int entinenIka = h.Ika;
h.Ika = 30;
```

Ominaisuudet

- set voidaan jättää määrittelemättä, jolloin ominaisuus (muuttujan arvo) voidaan vain lukea luokan ulkopuolelta
- set-metodissa voidaan ennen sijoitusta myös testata, onko tuotu arvo kelvollinen ja tuottaa poikkeus, jos ei ole
- Myös get voidaan jättää määrittelemättä, jolloin ominaisuuden (muuttujan) arvo voidaan vain asettaa luokan ulkopuolelta (käyttötarkoitus?)

Ominaisuudet: auto implemented properties

- Jos set ja get-ominaisuuksien sisältönä on vain arvon asettaminen ja palauttaminen, itse muuttujan nimi voidaan jättää kokonaan kirjoittamatta
- Esim. **`public String Nimi { set; get; }`**
- Ominaisuutta vastaavan muuttujan esittely ei siis näy koodissa lainkaan
- Jos setin ja getin pitää sisältää mitään muuta toiminnallisuutta, muuttujan nimi täytyy kirjoittaa näkyviin ja kirjoittaa ominaisuuksien määrittelyt auki edellisen esimerkin mukaisesti

Esimerkki 4-2 Opintojakso-luokka käyttämällä ominaisuuksia

```
public class Opintojakso
{
    private int opintopisteet;

    public String Nimi { set; get; }

    public Opintojakso(String n, int p)
    {
        Nimi = n;
        Pisteet = p;
    }
```

```
    public int Pisteet
    {
        get
        {
            return opintopisteet;
        }
        set
        {
            if (value > 0)
            {
                opintopisteet = value;
            }
        }
    }
}
```

Suojausmääreet

- Luokan jäsenen suojausmääre voi olla:
  - private – oletusarvo, private-jäsenet näkyvät vain luokan omissa jäsenmetodeissa
  - protected – näkyvät vain luokan ja sen aliluokan jäsenmetodeissa
  - internal – näkyvät vain samassa projektissa määriteltyjen luokkien jäsenmetodeissa
  - protected internal = proctected + internal -näkyvyys
  - public – julkinen näkyvyys, näkyvät kaikkialla, missä olioon on viite
- Luokan suojausmääre voi olla vain joko internal (oletus) tai public

Suojausmääreet

- public-tyyppisiksi tulisi määritellä
  - luokan rakentajat
  - luokan julkiseen rajapintaan kuuluvat metodit eli metodit, joita muut luokat tarvitsevat luokan palvelujen hyödyntämiseksi
  - Vakiot (const)

Suojausmääreet

- protected-tyyppisiksi tulisi määritellä:
  - sellaiset metodit, jotka eivät ole tarpeellisia luokan hyväksikäyttäjälle, mutta tarpeellisia sellaisille, jotka tekevät luokalle aliluokan
  - jos tietylle metodille halutaan erilainen toteutus aliluokkiin, se voidaan määritellä protected tai public –tyyppiseksi
    - Private-tyyppiset ominaisuudet eivät periydy
  - Protected käytännössä harvinainen

Suojausmääreet

- private-tyyppiseksi tulisi määritellä:
  - Kaikki data eli jäsenmuuttujat eli attribuutit
  - Kaikki muut kuin edellä mainitut metodit eli funktiot
- Koska data on aina yksityistä eli siihen voidaan viitata vain luokan itsensä sisältä, meidän tulee määritellä julkiset set- ja get-operaatiot sellaiselle datalle, jonka arvoja pitää voida lukea tai muuttaa toisista luokista käsin

Vakiot

- Vakioarvoja voidaan määritellä const-määreellä
  - Arvo on annettava muuttujan esittelyn yhteydessä

```
public const double PII = 3.14;
```

- Vain luettavaksi tarkoitettuja muuttujia voi esitellä myös readonly-määreellä
  - Readonly-muuttujalle voidaan antaa arvo sen esittelyn yhteydessä (vrt. const) **tai** luokan rakentajassa
  - Readonly-mahdollistaa arvon muuttamisen vielä ajon aikana, kunnes rakentajametodi on kokonaan suoritettu
  - Esim.

```
public readonly int i = 10;
```

Harjoitus

- Muokkaa Summa-luokkaa niin, että
  - luvut kysytään rakentajametodissa
  - luvut tallennetaan jäsenmuuttujiin (käytä mahdollisesti ominaisuuksia)
  - lukujen summa lasketaan erillisessä metodissa
- Noudata tätä tyyliä myös jatkossa!