C#:n oliomalli: luokat ja rajapinnat

Olio-ohjelmointi

Lapin AMK / EM

C#:n oliomalli

- Oliolla tarkoitetaan luokan ilmentymää
- C#-ohjelma koostuu luokista, struktuureista ja rajapinnoista
- Luokat voivat sisältää jäsenmuuttujia, ominaisuuksia ja metodeja
  - Jäsenmuuttujia nimitetään myös attribuuteiksi tai kentiksi
  - Metodeja nimitetään myös jäsenfunktioiksi tai operaatioiksi
- Jokainen metodi kuuluu johonkin luokkaan, metodeja tai muuta suoritettavaa koodia ei voi esiintyä luokkien ulkopuolella
- C#-lähdekooditiedostojen tarkennin on .cs

Esimerkki 6-1 Pet

```
namespace Pet
{
   public class Pet
   {
      private static int amount = 0;
      public String Name { set; get; }

      public Pet() : this("N/A")
      { }

      public Pet(String Name)
      {
         Pet.amount++;
         this.Name = Name;    // Huonoa tyyliä
      }
   }
}
```

Olion itseviite

- Olioon itseensä voidaan viitata sanalla **`this`**
- Olion itseviitettä tarvitaan, kun
  - kutsutaan toista saman luokan rakentajaa:
    `this("N/A")`
  - viitataan varjostettuun muuttujaan:
    `this.Name = Name;`
  - halutaan välittää koko olio funktion parametrina tai tallentaa olio taulukkoon tai muuhun säiliöön

Muuttujien varjostaminen (variable shadowing)

- Muuttujien varjostamisella tarkoitetaan sitä, että metodin parametrille tai paikalliselle muuttujalle annetaan sama nimi kuin luokan jäsenmuuttujalle
  - Tällöin paikallinen muuttuja peittää luokan jäsenmuuttujan siinä lohkossa, jossa se on määritelty
  - Jäsenmuuttujaan voidaan silti edelleen viitata laittamalla muuttujan eteen olion itseviite ***this***
- Varjostamista näkee käytettävän, mutta se ei ole yleensä suositeltavaa

Jäsenmuuttujien nimeäminen

- Eräs tapa välttää muuttujien varjostamisongelma on liittää luokan jäsenmuuttujien alkuun **m_** tai **a_** tai vain **_**
  - m on lyhenne sanasta member
  - a on lyhenne sanasta attribute
  - Näistä a on suositeltavampi, koska m-kirjaimen voidaan tulkita (virheellisesti) viittavan myös method-sanaan
  - Esim. private int a_size;
- Metodien parametreille ja paikallisille muuttujille voi tämän jälkeen käyttää samaa nimeä ilman etuliitettä
- Ominaisuuksien osalta ongelman voi välttää aloittamalla julkiset ominaisuudet isolla kirjaimella ja parametrit pienellä

Yliluokkaviite

- Yliluokkaan voidaan viitata sanalla ***super***

```
public class Dog : Pet
{
    // kutsutaan yliluokan rakentajaa
    public Dog(String name): super(name)
    {
        // Dog-luokan rakentajan oma koodi
    }
}
```

Olio- ja luokkamuuttujat

- Oliomuuttujista (instance variables) luodaan oma kopio jokaista luokan ilmentymää eli olioita kohden
- Oliomuuttujat alustetaan yleensä, kun olio luodaan. Alustus tapahtuu luokan rakentajassa ja arvot annetaan usein rakentajan parametreina:
  - `Pet myCat = new Pet("Repe");`

Olio- ja luokkamuuttujat

- Luokkamuuttujista eli staattisista muuttujista on olemassa vain yksi kopio, jonka luokan kaikki ilmentymät jakavat
- Jäsenmuuttuja määritellään staattiseksi **`static`**-määrellä
- Luokkamuuttujiin viitataan luokan nimen avulla:
  - **`Pet.amount++;`**
- Luokkamuuttujat alustetaan, kun luokka ladataan muistiin
  - Alustaminen voidaan tehdä joko muuttujan määrittelyn yhteydessä:
    - **`public static int amount = 0;`**
  - tai luokan ns. staattisessa alustusosassa:
    - static { amount =0; }
  - Arvoa voidaan muuttaa myös rakentajametodissa, jolloin se muuttuu aina, kun luokasta luodaan uusi olio. Näin on tehty esim. yllä olevan esimerkin lemmikkilaskurissa: Pet.amount++;

Olio- ja luokkametodit

- Olion ominaisuuksiin viitataan pistenotaatiolla eli olion nimellä ja pisteellä:
  - **`String s = myCat.Name;`**
- Yllä olevan metodikutsulla haetaan myCat-kissan nimi
- Samalla tavalla voidaan viitata myös jäsenmuuttujiin ja metodeihin

Staattiset eli luokkametodit

- Staattisia metodeja kutsutaan luokan nimen avulla:
  - `int lukumaara = Pet.getAmount();`
- Usein tarvittuja luokkametodeja ovat esimerkiksi Math-luokan sisältämät matemaattiset funktiot, esim.
  - `double luku = Math.Round(desimluku,1);`
    - **Pyöristetään luku yhden desimaalin tarkkuuteen**

Metodien kuormitus

- Metodien kuormituksella (overloading) tarkoitetaan sitä, että samassa luokassa on useita samannimisiä metodeja, jotka eroavat toisistaan **argumenttien tyypin ja/tai määrän suhteen**
- Pelkkä metodien paluuarvojen erilaisuus ei riitä
  - Jos metodit samassa luokassa eroavat toisistaan vain paluuarvoltaan seurauksena on käännösvirhe
- Esimerkin 6-1 Pet-luokalla on kaksi kuormitettua rakentajametodia
- On hyvä muistaa, että metodien kuormitus (overloading) on eri asia kuin perintään liittyvä metodien korvaaminen (overriding)

Periytyminen (inheritance)

- Perintä ilmaistaan kaksoispisteellä luokan otsikossa

```
class Dog : Pet
```

- Perinnässä aliluokka saa käyttöönsä kaikki yliluokkansa public ja protected -tyyppiset muuttujat ja metodit
- Private-tyyppiset ominaisuudet eivät periydy

Luokkahierarkia

- C#-luokkahierarkian juuriluokka on System.Object, jolla ei ole yliluokkaa
- Kaikilla muilla luokilla on aina täsmälleen yksi suora yliluokka, joka on oletusarvoisesti System.Object
- Jokainen luokka perii joko suoraan tai epäsuoraan toisten luokkien välityksellä System.Object-luokan

Periytyminen (inheritance)

- Aliluokka perii kaikki yliluokan public- ja protected -tyyppiset muuttujat sekä metodit
  - Voidaan käyttää ja kutsua kuin luokan omia muuttujia ja metodeja
- Aliluokkaan määritelty samanniminen muuttuja piilottaa yliluokassa olevan muuttujan
  - Muuttujien piilottaminen ei ole hyvää ohjelmointityyliä
  - Ongelma voidaan välttää kokonaan, kun data määritellään aina private-tyyppiseksi

Metodien korvaaminen

- Kun aliluokkaan määritellään samanniminen, paluuarvoltaan samanlainen ja samat argumentit saava metodi kuin yliluokkaan, metodi korvaa yliluokassa olevan metodin
  - Metodin otsikon tulee siis olla täsmälleen sama
  - Metodin suojausmääreen saa kuitenkin muuttaa vähemmän rajoittavaksi esim. protected -> public, mutta ei toisinpäin
    - Syy tähän on siinä, että kaikkialla missä voidaan käyttää yliluokan tyyppistä olioita, pitää voida käyttää myös aliluokan tyyppistä oliota
- Oliohierarkia sisältää siis useampia saman metodin toteutuksia
  - Myöhäisessä sidonnassa suoritettava metodi valitaan ajonaikaisesti kutsun tekevän olion tyypin mukaan
  - Ks. erillinen esimerkki Shape

Abstrakti luokka

- Abstrakti luokka (abstract class) on vain periytettäväksi tarkoitettu luokka, josta itsestään ei voi luoda ilmentymää
- On kuitenkin mahdollista määritellä abstraktin luokan tyyppisiä muuttujia. Minkä tyyppisiin olioihin niillä voi viitata?
- Abstrakti metodi on metodi, jolle on annettu nimi, argumentit ja paluuarvo, muttei toteutusta
  - Annetaan siis metodin otsikko (header, prototype, signature) ilman runkoa (body)
- Luokka on abstrakti, jos se sisältää yhdenkin abstraktin metodin

Rajapinnat

- Rajapinnat voivat sisältää **vain** vakioita tai metodien otsikoita
- Rajapinnat muistuttavat abstrakteja luokkia. Ne eivät kuitenkaan voi sisältää lainkaan metodien toteutuksia (”puhdas abstrakti luokka”)
- Luokka voi toteuttaa yhden tai useamman rajapinnan tai olla toteuttamatta yhtään
  - Rajapinnan toteuttavan luokan on sisällettävä toteutus jokaiselle rajapinnan metodille

Rajapinnat

- Toteutettavat rajapinnat kerrotaan luokan otsikossa perittävän luokan jälkeen pilkulla erotettuna. Esimerkki:

```
class Dog : Pet, IComparable
```

- Rajapinnasta ei voi luoda ilmentymää eli oliota
- Rajapintatyyppisiä muuttujia voi kuitenkin esitellä aivan samaan tapaan kuin luokkatyyppisiä muuttujia
- Rajapinta-tyyppisellä muuttujalla voi viitata olioihin, jotka toteuttavat kyseisen rajapinnan

Abstrakti luokka ja rajapinta

- Abstrakti luokka voi sisältää abstraktien metodien lisäksi myös metodien toteutuksia ja muuttujia (kuten tavallinen, konkreettinen luokka)
- Rajapinta (interface) voi sen sijaan sisältää **vain** abstrakteja metodeja ja vakiomäärittelyitä
  - ”puhdas abstrakti luokka”
- Rajapinnan hyödyntäjän tehtävä on antaa toteutus kaikille rajapinnan metodeille

Monimuotoisuus (polymorphism)

1. Samaa muuttujaa voidaan käyttää viittaamaan useampien, periytymishierarkiassa eri tasolla olevien luokkien ilmentymiin
   - Minkätyyppistä muuttujaa voidaan käyttää viittaamaan periytymis-hierarkiassa eri tasolla olevien luokkien ilmentymiin?
2. Samalla operaatiolla voi olla luokkahierarkiassa monta eri toteutusta (metodien korvaaminen)
   - Miten?

Monimuotoisuus

- Vastaukset:
  1. Yliluokka-tyyppistä osoitinmuuttujaa tai viitemuuttujaa voidaan käyttää viittaamaan myös aliluokkien ilmentymiin
  2. Aliluokat voivat korvata yliluokkansa virtuaalisen operaation (=antaa sille oman toteutuksen)

Aikainen ja myöhäinen sidonta

- Aikainen eli staattinen sidonta (early binding, static binding): operaation kutsu liitetään jo käännösaikana kutsuttavan operaation määrittelyyn
- Myöhäinen eli dynaaminen sidonta (late binding, dynamic binding): funktiosidos syntyy vasta kutsuhetkellä; kutsuttava aliohjelma määräytyy vasta **ajon aikana** kutsuvan olion tyypin (luokan) mukaan
- Myöhäinen sidonta on monimuotoisuuden toteutusmekanismi ja siten olio-kielten keskeinen ominaisuus
- Myöhäistä sidontaa voidaan soveltaa vain oliometodeihin, ei luokkametodeihin
  - Miksi?

Esimerkki myöhäisestä sidonnasta

```
public class Test
{
  public static void Main(string[] args)
  {
    // Shape is the superclass of the Circle, Rectangle and
   Triangle
    Shape s;

    s = new Circle(5);
    Console.WriteLine("The area is {0}", s.getArea());

    s = new Rectangle(7,3);
    Console.WriteLine("The area is {0}", s.getArea());

    s = new Triangle(17,7,Math.PI/3);
    Console.WriteLine("The area is {0}", s.getArea());
  }
}
```

Virtuaalioperaatio

- Operaatiota, johon myöhäistä sidontaa voi soveltaa kutsutaan virtuaaliseksi operaatioksi
  - C#:ssa ja C++:ssa ja operaatio ei ole virtuaalinen, ellei sitä nimenomaisesti määritellä sellaiseksi virtual-määreellä
    - Korvattavalle metodille tulee aliluokassa antaa lisäksi C#:ssa override-määre
    - Mitä tapahtuu, jos myöhäistä sidontaa yritetään soveltaa ei-virtuaaliseen metodiin?
  - Javassa operaatio on aina virtuaalinen ellei metodin määrittelemistä uudelleen aliluokissa nimenomaisesti kielletä final-määreellä

Harjoitus

- Selvitä itsellesi Kuvio (Shape) esimerkin toiminnallisuus
- Lisää Shape-luokalle uusi aliluokka Suunnikas (Parallelogram). Pinta-ala = sivu1 * sivu2 * sin(sivujen välinen kulma)
  - Sini-funktio: Math.sin(kulma radiaaneina)
  - Asteet radiaaneiksi: Math.PI / 180 * kulma asteina
  - Pyöristys-funktio: Math.Round(luku)
- Luo tämän tyyppinen olio Test-luokassa ja kutsu printAreaAndCircumference-metodia