Poikkeusten käsittely

Olio-ohjelmointi

Lapin AMK / EM

Poikkeusten käsittely: try-catch-finally

- C#:ssa poikkeusten käsittelyyn käytetään try-catch-finally –rakennetta ja Exception- eli poikkeusluokkia
- Poikkeustilanteissa metodeista ei poistuta return-lauseella, vaan nostamalla throw-komennolla poikkeus
- Kun metodi nostaa poikkeuksen, se luo ilmentymän jostakin poikkeusluokasta
  - System.Exception on kaikkien poikkeusluokkien kantaluokka
  - Siitä perittyjä luokkia ovat esimerkiksi ArithmeticException, IndexOutOfRangeException ja NullReferenceException
- Jos metodi voi nostaa poikkeuksia, sitä kutsuvan metodin on varauduttava käsittelemään poikkeukset try-catch –rakenteessa
  - Kun kutsut luokkakirjaston metodia, tarkista dokumentaatiosta, mitä poikkeuksia se voi aiheuttaa

Poikkeusten käsittely: try-catch-finally

- Esimerkki try-catch –lohkosta:

```
float osamaara;
try
{
  osamaara = jaettava/jakaja;
}
catch (DivideByZeroException e)
{
  Console.WriteLine(”Jako nollalla”);
}
```

Poikkeusten käsittely

- try-lohkossa kerrotaan normaali suoritussekvenssi
- Mikäli try-lohkossa tapahtuu virhe, suoritus siirtyy catch-lohkoon
- catch-lohkoja voi olla useita ja niistä suoritetaan se, joka ensimmäisenä vastaa virheolion tai sen yliluokan tyyppiä
- Jos mikään catch-lohko ei vastaa try-lohkossa tapahtunutta virhettä suoritus keskeytyy ajonaikaiseen virheeseen
- finally-lohko on vapaaehtoinen, yleensä jää pois

Omat poikkeusluokat

- C#:ssa voidaan myös määritellä omia poikkeusluokkia
- Omien poikkeusluokkien tulee periä Exception-luokka
- Itsemääriteltyihin virheluokkiin voidaan liittää attribuutteja, jotka kertovat lisätietoja virheestä, esim. mihin olioon virhe kohdistui. Lisäksi Exception-luokan Message-metodi voidaan korvata omalla toteutuksella

Omat poikkeusluokat

```
public class AsiakasPoikkeus : Exception
{
  public long AsiakasID;
  public AsiakasPoikkeus(string message, Exception
  e, long AsID): base (message, e)
  {
    AsiakasID = AsID;
  }
  public override string Message
  {
    get {
      return ”ID:”+AsiakasID+” ”+base.Message;
    }
  }
}
```

Resurssien vapautus

- Virhetilanteissa resursseja voi jäädä vapauttamatta, kun try-lohkon suoritus keskeytyy ja suoritus siirtyy catch-lohkoon
- Ongelma voidaan ratkaista sijoittamalla resurssien vapautus finally-lohkoon, joka suoritetaan aina tuli try-lohkossa virhettä tai ei. Esim.

```
finally {
 if (streamWriter != null)
  streamWriter.close()
}
```

Resurssien vapautus

- Resurssien vapauttamiseksi finally-lohkossa on olemassa on lyhennysmerkintä:

```
using (obj) { ...
}
```

- joka laventuu koodiksi:

```
try {
}
finally {
  if (obj != null)
    obj.Dispose();
}
```

- Using-lauseen sisällä viitattavan tai luotavan olion tulee toteuttaa IDisposable-rajapinta
  - Rajapinta sisältää metodin void Dispose(), jonka toteutuksessa resurssit vapautetaan

Resurssien vapautus

- Esimerkki:

```
StreamWriter sw;
using (sw = new StreamWriter(
 File.Open(tiedostonimi, FileMode.Create)))
 {
  try {
    sw.Write(”Kirjoitetaan tiedostoon”);
  }
  catch (Exception e) {
    // Virheenkäsittely
  }
} // using
```

Kommentointi

- C#-koodiin voidaan kirjoittaa kommentteja kolmella eri tavalla:
- /* merkeillä alkava kommentti jatkuu, kunnes tulee kommentin päättävä */
- // merkit aloittavat yksirivisen kommentin
- /// merkit aloittavat yksirivisen xml-kommentin

XML-kommentit

- XML-kommentit alkavat /// -merkillä
  - Nämä rivit lisätään xml-tiedostoon
  - Voit tehdä kyseisen xml-tiedoston komentorivillä komennolla:

    csc ForminNimi.cs /doc:xmlDoc.xml
    - Saman voi tehdä myös Visual Studiolla

XML-kommentit

- Xml:ää vaikea lukea -> olisi hyvä saada esim. html-tiedosto
  - Tee tämä Visual Studiolla
    - Tools->Build Comment Web Report
  - Lisätietoja kommentoinnista voit löytää C# Programmers Referencestä, joka on sisällytetty Visual Studioon