Repetition af Objekt Orienteret Programmering 2.HF

It & Data, Odense

ANNL/april 2019

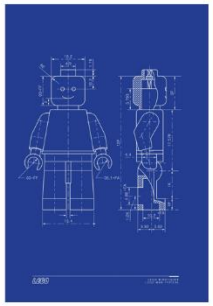
SYDDANSK ERHVERVSSKOLE

Øvelse:
Den varme stol

Spørgsmål til den varme stol
Opsummering af del 1 (H1)

- Hvad er et objekt?

Class



Instantiering

Objekt

- Hvorfor OOP?
- Hvad er incapsulation?
  - Access modifiers?
    - Public, private, protected, internal (Assembly)
- Klassens bestanddele:
  - Constructor / Destructor?
  - Fields vs. properties?
- Hvad er nedarvning?
- Hvad er polymorfi?

Klassens bestanddele

- En **klasse** i C# har altid et **Navn**. Vi navngiver klassen ud fra hvilket objekt, der kan instantieres af klassen *eks. LegoMan.* Klassenavnet begynder altid med **S**tort og anvender **C**amel**C**ase.

- Klasser kan relatere til hinanden på flere forskellige måder: Nedarvning, interfaces, abstraktion – de forskellige relationer gennemgås senere.

- Klassen kan have en **Constructor**. En **Constructor**, er en metode, der automatisk kaldes ved instantiering af et nyt objekt af klassen.

- Klassen kan have en **Destructor**. En **Destructor**, er en metode, der automatisk kaldes ved et instantieret **objekts ophør** med at eksisterer. En destructor bruges typisk til at afbryde evt. dataforbindelser, lukke en åben fil eller lignende ”oprydning”, der ikke håndteres af .NET frameworkets garbage collector.

- Klassen kan have **Fields** (klasse variabler), typisk Private.

- Klassen kan have **Properties**
  (med get og/eller set metoder).

- **Demoeksempel:** *LegoMan klassen*

Indkapsling (incapsulation)

- Et **objekt** er en **instans** af en **klasse**, og klassen er grundlæggende et lille selvstændigt og lukket program.
  Det skal forstås på den måde, at klassens bestanddele, som udgangspunkt, kun kan tilgås fra klassen selv.

- Det er udvikleren af klassen, der definerer, hvordan vi kan tilgå klassens bestanddele udefra. På denne måde, sikre vi vores kode fra at blive anvendt forkert og dermed brække programmet.

- **Indkapsling** er en **grundsten** i **OOP**.

Access modifiers (Specifiers)

- **Adgangen** og anvendelsen af en klasses **fields**, **properties** og **metoder** kan **kontrolleres** med såkaldte **Access Modifiers**.

- I **C#** findes der **4** forskellige **Access Modifiers**:
  Public, **Private**, **Protected** og **Internal**.

- Vi angiver vores **Access Modifiers** i deklarationen af henholdsvis selve klassen og klassens: fields, properties og metoder.

- **Public** – kan tilgås direkte på et objekt af klassen, også udefra.
- **Private** – kan KUN tilgås i klassen selv.
- **Protected** – kan tilgås fra klassen og fra nedarvede klasser.
- **Internal** – kan tilgås fra samme assembly (exe, dll).

Nedarvning

- En **klasse** kan **nedarves** fra en anden klasse. På denne måde, arver en klasse al funktionalitet fra ovenstående klasser. Med nedarvning sparer vi kode og dermed tid.

- I en klasse der arver en ovenstående klasse, kan vi:
  - Genbruge arvet funktionalitet
  - Udvide arvet funktionalitet
  - Overskrive/ændre arvet funktionalitet

- Nedarvning giver dynamik til OOP og gør det nemmere at udvikle, vedligeholde og genbruge.

- Find fællesnævnerne for objekter og overvej om der er grubund for nedarvning.

- Eks. Forestil dig bil, cykel, lastbil, gravko objekter!
  Er der noget de har til fælles?

Polymorfi

- **Polymorfi** er et **begreb**, der findes i **OOP** og som betyder at **klasser** kan have **forskellig** funktionalitet, selvom de deler samme **interface**.

- I praksis betyder det: at to forskellige klasser, der **arver** fra samme ovenstående klasse, kan have forskellig funktionalitet/implementering af den samme metode (metode med samme navn, men forskelligt indhold).

- **Eks.** Forestil dig klassen **køretøj**.

  køretøj definerer (ingen implementering) en **metode** der hedder: ***Brems().***

  Køretøj **nedarves** til klassen: **Bil** og klassen **Cykel**.

  Både klassen **Bil** og klassen **Cykel, overskriver** den arvede metode **Brems()**, men implementerer hver sin måde at bremse på.

  - Bil -> Brems() -> *"træd på bremsen"*
  - Cykel -> Brems() -> *"træk i håndbremsen"*

Collections (gruppering af objekter)

En mere dynamisk måde at opbevare data på end arrays

Klasse	Beskrivelse
List <T>	Standardliste, som kan sammenlignes med et array men med flere muligheder
Dictionary<TKey,TValue>	En liste, som opbevarer data i en nøgle-/værdi-struktur
Query<T>	En liste, der kan benyttes som en traditionel kø (FIFO)
Stack<T>	En liste, der kan anvendes som en traditionel stak (LIFO)
SortedList<TKey,TValue>	En liste, som opbevarer data i en nøgle-/værdi-struktur, der er sorteret efter nøglen. Værdier hentes via nøgle eller indeks.

<T> står for type (datatype – såsom string, int, double osv.)

Exception handling (Fejlhåndtering)

Det er umuligt at skrive kode, som ikke kan fejle på en eller anden måde.

Enten på grund af decideret kode-fejl eller en uforudset opstået fejl.

Kode til at håndtere fejl:

- Try-catch
- Try-cach-finally

```
try
{
    using (StreamReader sr = File.OpenText("data.txt"))
    {
        Console.WriteLine($"The first line of this file is {sr.ReadLine()}");
    }
}
catch (FileNotFoundException e)
{
    Console.WriteLine($"The file was not found: '{e}'");
}
catch (DirectoryNotFoundException e)
{
    Console.WriteLine($"The directory was not found: '{e}'");
}
catch (IOException e)
{
    Console.WriteLine($"The file could not be opened: '{e}'");
}
```

Throw:

Husk at sende fejlen videre i systemet, så slutbrugeren får besked.

Egne exceptions:

Mulighed for at angive en fejl, hvis ens egne regler ikke er overholdt.

Interfaces (recap)

- Hvad er en Interface klasse?
  - En **kontrakt** mellem Interface klasser og implementerede klasser.
  - En klasse, der implementerer et Interface (Implements), tvinges til at implementerer metoder, der er defineret i Interface klassen.

- Hvorfor interfaces?
  - Vedligeholdelse af kode
    - Best practise: Programmer til en abstraktion (Interface/kontrakt) frem for en konkret type (Collections).
  - Udvidelse af funktionalitet
  - Testbart

Abstrakte klasser

- En abstrakt klasse minder om et interface, men til forskel for Interfacet er det en klasse.
- Der kan **ikke** instantieres objekter af abstrakte klasser, den kan kun arves af underliggende klasser.
- En abstrakt klasse kan have både abstrakte metoder, der ikke har nogen implementeret kode og reelle metoder med færdig implementeret kode.
- En klasse, der arver en abstrakt klasse, tvinges gennem en ”kontrakt” til at færdiggøre (implementere) de abstrakte metoder.
- En abstrakt klasse kan bruges til at sikre, at underliggende klasser, overholder de angivne metoder. Samtidig kan underliggende klasser gøre brug af den funktionalitet, der stilles til rådighed gennem de almindelige metoder.

Abstraktion af klasser

- Sammenligning af Interface, Abstract og alm. klasser (Concrete)

	Der kan Inst. objekter af klassen	Compiler error ved mangl. Implemetrering af metoder i nedarvet/implements klasser	Kan nedarves	Kan indeholde implementeret metoder	Kan indeholde IKKE implementerede metoder (Deklareret metoder)	Multiple arv	Kan anvende Access Modifiers
Klasse (Concrete)	X		X	X			X
Abstrakt klasse		X	X	X	X		X
Interface klasse		X	X (Implements)		X	X En child klasse kan implementerer flere interfaces og arve fra en klasse på samme tid	Altid public

Kan indeholde:	Klasse	Abstrakt klasse	Interface klasse
Fields	X	X	
Properties	X	X	X
Methods	X	X	X
Constructor	X	X	
Destructor	X	X	
Events	X	X	X
Indexers	X	X	X

Slut på OOP repetition

Hvis der er spørgsmål til emnet, er det nu, de skal på banen...