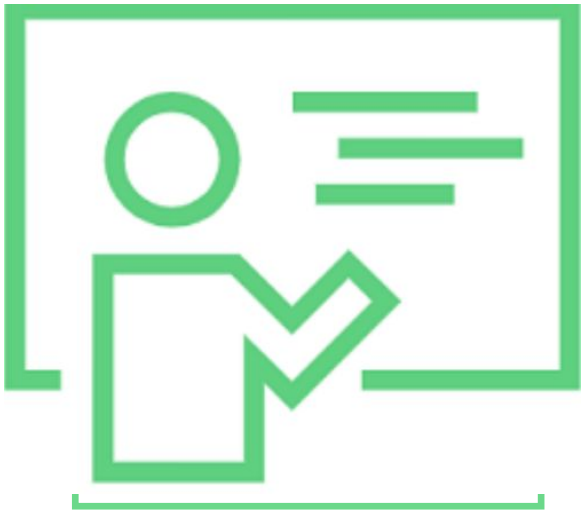

Grundkurs i C# och .NET Framework

Kurs för NFI DEL3

Stefan Holmberg, Systementor AB

OOP

- Objektorienterad programmering
  - Paradigm
    - cirka 1960-talet (Simula 67)
  - Ett sätt att tänka
  - Systemdesign / arkitektur utifrån *objekt*
  - Återanvändbar kod+data(state) genom *objekt*
- OBS: OOP är *inte* en viss typ av programmeringsspråk
  - det är ett sätt att tänka, lösa / bryta ner problem

OOP: modularisering

- Lättare att skriva och underhålla kod om man:
  - Delar upp det i mindre bitar
  - Bygga en bit i taget
- -> Modularisering
- Dela upp applikationen, och bygga/testa varje del för sig
- Sätta sedan ihop alla delar
- Lättare att skriva tio små program
  - än att skriva en stor (10x) program

OOP: modularisering

- Om modularisering ska fungera
  - 🌕 Så måste modulerna vara skilda från varandra,
  - 🌕 De ska gömma information från andra moduler,
  - 🌕 Men de måste också ge ett gränssnitt utåt
- -> Objekt
- = svart låda med DATA och FUNKTIONALITET – som användare av objektet struntar vi i innanmätet (Random.Next(...))

Vad är en klass ?

- En klass är en mall som man bygger upp kring ett informationsobjekt som man hanterar. Det är alltså något man kan hantera information om tex en kund, order, produkt.

- Den hanterar både informationen och funktionalitet kring informationen.

- När man skall använda en klass skapar man en sk instans av klassen, man kan säga att det är en kopia av mallen som man använder. En instans av klassen kallas även för ett objekt. Därifrån kommer begreppet objektorienterad programmering

OOP: Objekt

- Vad är ett objekt..?
- Ett sätt att modularisera (paketera) kod
- Speglar konceptet av objekt i "verkligheten" (modellen)
- Objekt har två egenskaper:
  - Tillstånd (state) - attribut / variabler
  - Beteende (behaviour) - metoder / funktioner

Egenskaper (properties)

- En klass har egenskaper. Det är som en variabel som lagrar värden som hör till klassen

- Alla egenskaper hanteras genom variabler som deklareras inne i klassen. Dessa variabler kallas för instansvariabler

- Exempel för SpelOmgång (GissaEttTal) kan vara att klassen ha egenskaper som tex
  - AntalGjordaForsok
  - HemligaTalet
  -

Metoder

För att kunna göra något med klassen och hantera de egenskaper som finns måste man skapa metoder i klassen. För GissaEttTal kan klassen Spel

- Ha en metod som heter Restart
- Ha en metod som heter Guess

Konstruktor

- Varje klass har en metod som alltid körs direkt när en instans av klassen skapas.
- Den anropas automatiskt. Alla andra metoder måste anropas genom kod.
- Denna metod kallas för Konstruktor och har samma namn som klassen. Tex är metoden med namnet Konto konstruktor för Kontoklassen
- Konstruktorn brukar användas för att sätta startvärden på egenskaper i klassen. Den kan inte returnera någonting.

Inkapsling

- Inkapsling är ett viktigt begrepp som används för att öka oberoendet mellan olika klasser.

- Det innebär att metoder och egenskaper kan vara privata eller publika. En privat metod kan man tex inte komma åt utanför klassen utan den används internt inom klassen av andra metoder. En publik metod är motsatsen dvs kan anropas från andra klasser.

NACKADEMIN

OOP: Klass

- Vad är en klass..?
  - En klass speglar ett koncept i programmet
  - Är också en användar-definierad typ
    - Där man själv definierar hur man skapar och använder instanser (objekt) av denna typ
  - En samlingsobjekt för
    - Variabler
    - Metoder (funktioner)
  - Byggsten i stora, komplexa program

OOP: Vad är skillnad på klass och objekt

- Klass = ”ritning/modell”
- Objekt = en sak som skapats utifrån ritningen

- Ex finns ritning på hus. Det är en klass/definition på hur huset ska se ut
- Sen kan MÅNGA verkliga hus (objekt) skapas utifrån denna ritning

OOP: Hur kan det bli olika objekt om samma ritning?

- Ritning – det finns 4 väggar. Alla väggar har en egen färg
- Olika hus(objekt) kan ju ha olika färg på sina väggar

- ”Variabler, state”

OOP: Klass

- Vad består en klass (ritning) av?

- Interface (??? = gränssnittsyta) med
  - Data (attribut / variabler)
  - Metoder (funktioner)
  - Constructor (speciell metod)
  - Destructor (speciell metod)
  - Access (private, protected, public)

OOP: Klass - metoder

- Publika (”interface”)
- Interna (”helpers”)
- Varianter på interna:
  - private = osynlig för ALLA utom för kod i egna klassen
  - protected = osynlig utifrån men tillgänglig för subklasser

Länkar

https://introprogramming.info/english-intro-csharp-book/read-online/chapter-20-object-oriented-programming-principles/

C#: Objektorientering – the need for inheritence

Vi bygger ett lönesystem!!!

1. Anställda har namn, kontonummer, månadslön, funktion för LoneKorning

2. Men typen säljare ska ha ev provision också! 2% på det som den sålt för

3. Timanställda har timlön. Så man måste hålla reda på antal arbetade timmar

4. Stoppa in i lista – kör Lonekorning

C#: Objektorientering

PDF Chapter 11 + Chapter 14

C#: Modifiers and Access Levels (Visibility)

PDF Sidan 508

Labbar

ENUMS

Vad vill vi undvika? “Magic strings”

if (player.Position == “forward”) etc etc

```
enum Position
{
    Goalie,
   Defence,
    Forward
}
```

The Framework

- Använd de inbyggda behållarklasserna i .NET Framework
  - ArrayList
  - Stack
  - Queue
  - List
  - Dictionary

Felhantering - runtime errors

- Är koden felaktigt skriven blir det fel när den kompileras.
- Det finns andra typer av fel som inte beror på att koden är felaktigt skriven utan att på att något blir fel under tiden som koden körs.
- Exempel på när det kan bli runtime error är om man försöker tilldela en variabel ett värde av en annan datatyp utan att konvertera, om man försöker dividera ett tal med 0 eller om kontakten med databas- eller webbservern försvunnit.
- Runtime errors måste hanteras i koden för att programmet inte skall avbrytas eller krascha.

Exempel runtime error meddelande

Try- Catch

Innebär att koden delas in i två block

Try blocket: Här skrivs all kod som körs under normala omständigheter dvs när det inte blir fel. Blir det fel avbyts det här och körningen hamnar i catch blocket. Blir det inte fel hamnar koden aldrig i catch blocket.

Catch blocket: Här fångas felet upp. Ofta hanteras alla fel på samma sätt, men ibland vill man kontrollera typen av fel och hantera olika beroende på vad det är som blir fel.

Exception klassen

- Generell felklass som hanterar alla typer av exceptions. System.Exception
- Alla andra felklasser ärver från denna klass.
- Exempel på andra klasser som ärver från Exception är tex FileNotFoundException och IndexOutOfRangeException
- Andra felklasser fångar upp mer specifika fel och inte som Exception som fångar alla fel.

Try-catch

Här är ett exempel. Med hjälp av klassen Exception går det att få fram vad det är för typ av fel och vilket felmeddelande det genererade.

```
try
{
	//Här körs koden

}
catch( Exception ex)
{
	/*Här hamnar körningen om något går fel. Detta exempel skriver	ut felmeddelandet på skärmen */
	Messagebox.Show(ex.Message);
}
```

Try-catch -finally

- Efter en try-catch går det att lägga till ett finally-block. Där läggs kod som körs oavsett om det blir fel eller inte.

- Detta kan vara användbart i vissa lägen, oftast om man vill stänga objekt som används i koden. Exempel på detta kan vara en filestream som man skriver till en fil med eller en uppkoppling mot en databas.

Try-catch -finally

```
Filestream myStream;

try
{
        myStream = new FileStream("c:\\streamtest.txt", true)
}
catch( Exception ex)
{
        /*Här hamnar körningen om något går fel.*/
}
finally
{
        /*Här stängs tex objekt som används i koden */
        myStream.Close();
}
```

Egenstudier

https://introprogramming.info/english-intro-csharp-book/read-online/chapter-12-exception-handling/