# Programmering C#

Finn Vilsbæk

finv@kea.dk

# Emner for i dag

- Operatorer
- Branching

- **Læs:** ***Learning C#, kapitel 4 og 5***

# Operatorer

- Et symbol ( +, =, > ) som får C# til at gøre noget bestemt
- Brugbart til at sammenligne to værdier i et branching statement
- Operatorer bruges ofte i *expressions* – ie. ethvert C# statement som returnerer en value.
- Assignment operator: =
- Equality operator: ==

# Operatorer

- C# relationelle operatorer – eksempler på statements

*Table 4-1. C# relational operators (assumes bigValue = 100 and smallValue = 50)*

| Name | Operator | Given this statement | The expression evaluates to |
|---|---|---|---|
| Equals | == | bigValue == 100<br>bigValue == 80 | True<br>False |
| Not equals | != | bigValue != 100<br>bigValue != 80 | False<br>True |
| Greater than | > | bigValue > smallValue | True |
| Greater than or equal to | >= | bigValue >= smallValue<br>smallValue >= bigValue | True<br>False |
| Less than | < | bigValue < smallValue | False |
| Less than or equal to | <= | smallValue <= bigValue<br>bigValue <= smallValue | True<br>False |

# Operatorer

- C# logiske operatorer – evalueringer for x = 5 og y =7

*Table 4-2. Logical operators*

| Name | Operator | Given this statement | The expression evaluates to | Logic |
|---|---|---|---|---|
| And | && | (x == 3) && (y == 7) | False | Both must be true. |
| Or | \|\| | (x == 3) \|\| (y == 7) | True | Either or both must be true. |
| Not | ! | !(x == 3) | True | Expression must be false. |

# Operatorer: øvelse

**Exercise 4-3.** Imagine an amusement park ride that holds two passengers. Because of safety restrictions, the combined weight of the two passengers must be more than 100 pounds, but no more than 300 pounds. Now imagine a family of four who want to ride this ride. Abby weighs 135 pounds, Bob weighs 175 pounds, their son Charlie weighs 55 pounds, and their daughter Dawn weighs 45 pounds.

Write a program that calculates whether the weight of the two combined passengers falls within the accepted range. Use constants for the maximum and minimum weights, and for the weight of each family member. The output should look something like this, for Abby and Dawn:

```
Abby and Dawn can ride? True
```

Calculate three separate cases: whether the two parents can ride together, just Bob and Charlie, and just the kids.

# Operatorer: øvelse

- Handout hints, øvelse 4-3 fra Learning C# bogen:

- bool canRide = ((weight1 + weight2) > minWeight) && ((weight1 + weight2) <= maxWeight);

- Console.WriteLine(”Abby and Bob can ride? {0}”, canRide);

# Branching

- Unconditional
- Conditional

- Lader dit program bestemme ved runtime, hvilke stykker kode der skal eksekveres.

- Lader dig evaluere brugerens input, og afgøre en 'desired course of action' baseret på brugerens input.

- Real world scenarie: et *branch statement* kan kalkulere hvornår der skal sælges eller beholdes aktier, baseret på den aktuelle kurs for den enkelte aktie.

# Branching

- Unconditional

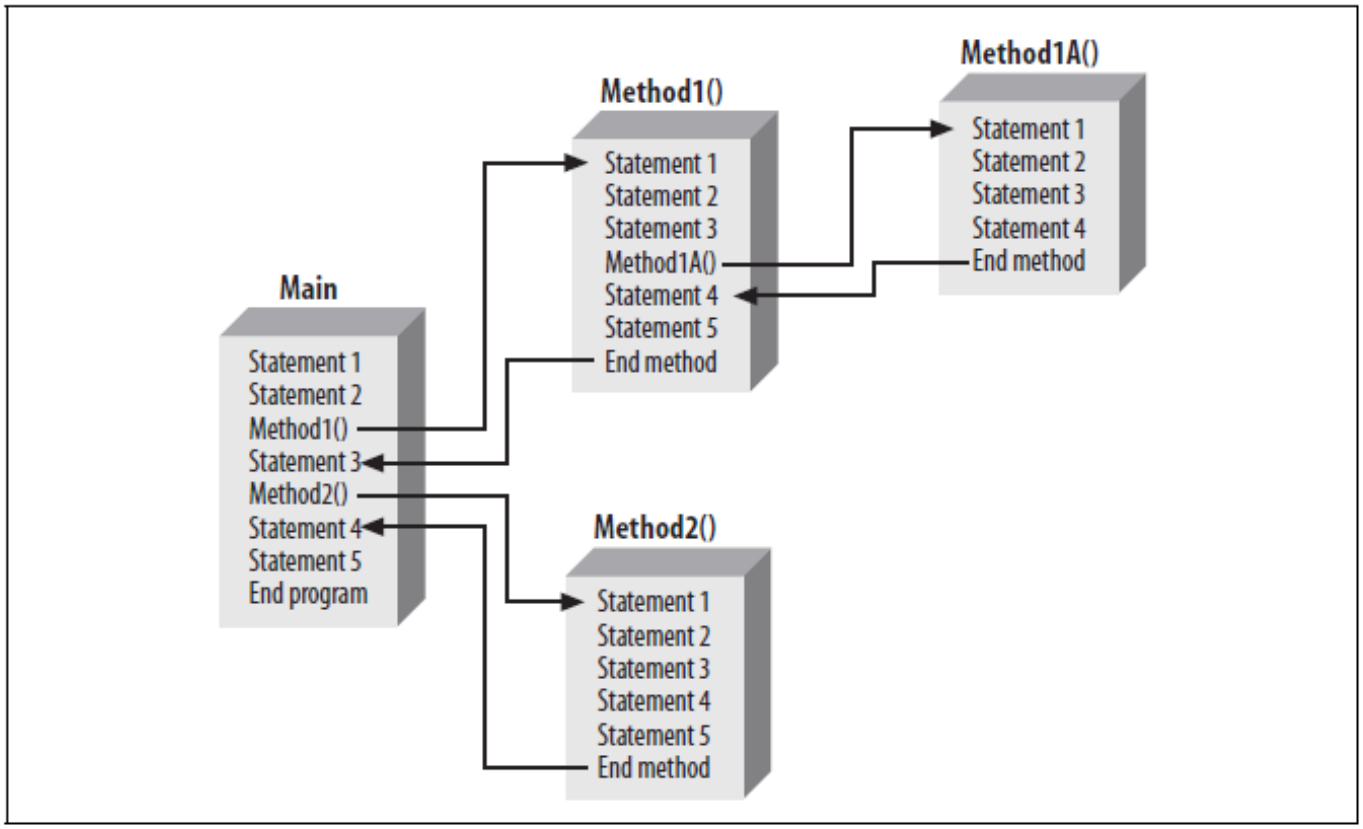
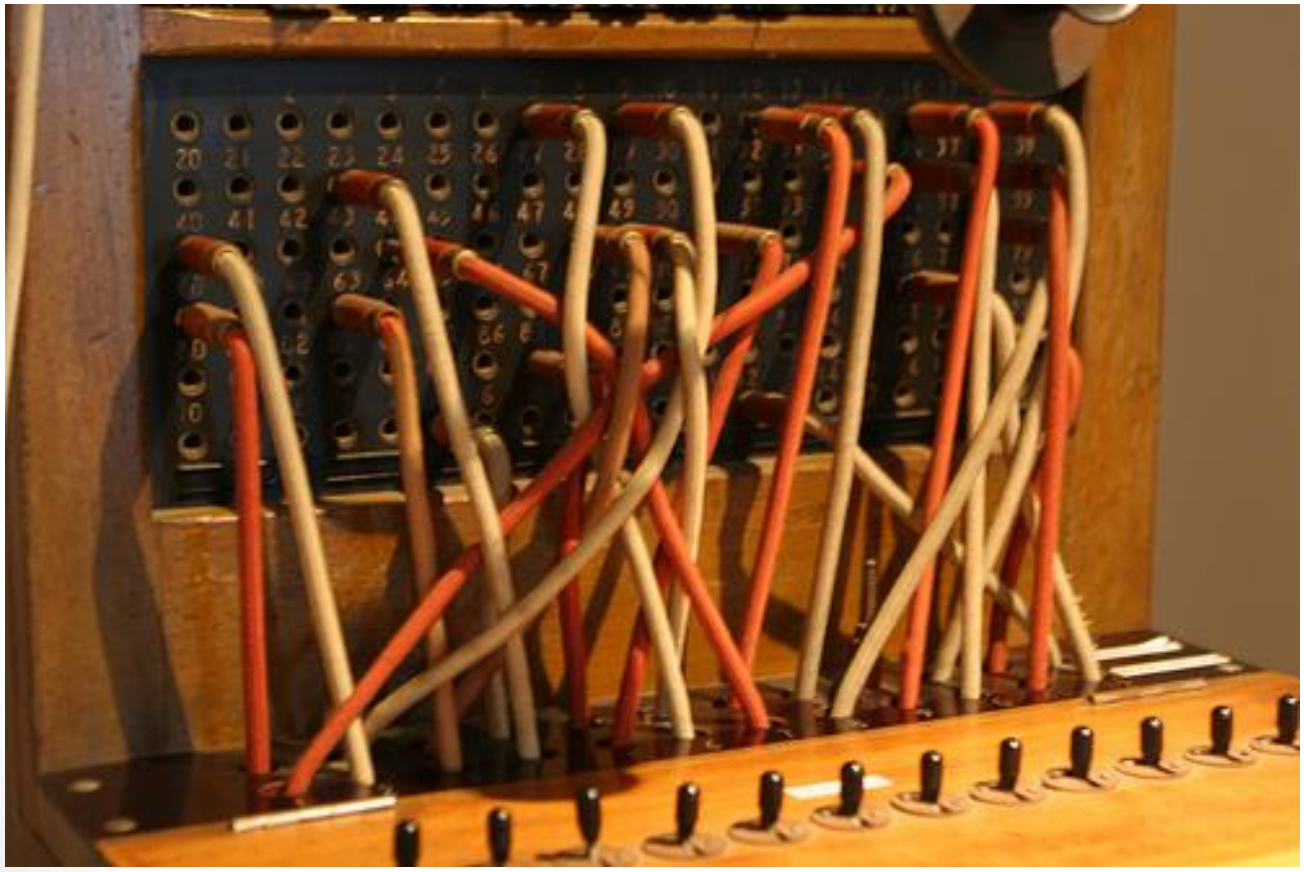



*Figure 5-1. Branching allows the execution to move around to different parts of your program, instead of simply proceeding in a straight line.*

# Branching

- Conditional branching i C#:  **switch** og **if-else** statements



# Branching

- If – then – else statements
- “Hvis en givet betingelse er sand, så eksekver det følgende statement; ellers spring over og gå videre i programmet.”
- Betingelsen er altid et boolsk udtryk:  true / false

# Branching – if statements

- Bruges til at teste 'edge cases' før en linie kode eksekveres
- AKA 'short-circuit evaluation'

```
public bool QuotientOverTwenty(float dividend, float divisor)
{
if (( divisor != 0 ) && ( dividend / divisor > 20 ))
        {
                return true;
        }
        return false;
}
```

# Branching

Nested if statements øvelse:

Lav en console app som:

- Prompter brugeren for input,
- accepterer en integer (et heltal) som input, og derefter
- evaluerer på om det heltal er lig nul,
- er et ulige eller lige tal,
- er et multiplum af 10, eller for stort (størrre end 100).

Brug flere forskellige niveauer (nested) af if statements.

# Branching

How to structure your work:

- 1) Test om tallet er for stort (større end 100).
- 2) Hvis det er, så stop programmet.
- 3) I den første if clause: check hvorvidt input er lige eller ulige.
- 4) Hvis input er ulige, så stop programmet.
- 5) I den anden if clause: check om input er et nul.
- 6) Hvis input er forskellig fra nul, så check herefter om det går op i 10.

Du skulle gerne ende med fire if niveauer, og en bunke lettere kompliceret kode.

# Branching

Du kan med fordel starte med at bygge programmet som følger:

```
static void Main(string[] args)
{
    while (true)
    {
        Console.Write("Enter a number please: ");
        string theEntry = Console.ReadLine();
        int theNumber = Convert.ToInt32(theEntry);
        // Logic: if the number is greater than 100, say it is too big
        // if it is even but not a multiple of 10 say it is even
        // if it is a multiple of 10, say so
        // if it is not even, say it is odd
        if (theNumber <= 100)
        {
            if (theNumber % 2 == 0)
            {
```

# Branching

- Lad os nu omskrive programlogikken til at gøre det samme arbejde, men  med brug af et *switch* statement istedet:

```
switch (expression)
{
    case constant-expression:
        statement
        jump-statement
    [default:
        statement]
}
```

# Branching

- Vi får brug for en 'condition' enum i den nye version af programmet (hint: lav en ny konsol-app):

```
class Program
{
    enum numericCondition
    {
        even,
        multiple,
        odd,
        tooBig,
        unknown,
        zero
    }
```

# Branching

- Du kan begynde sådan her efter at have deklareret  din enum:

```
static void Main(string[] args)
{
    while (true)
    {
        Console.Write("Enter a number, please: ");
        string theEntry = Console.ReadLine();
        int theNumber = Convert.ToInt32(theEntry);
        numericCondition condition = numericCondition.unknown; // initialize

        condition = (theNumber % 2 == 0) ? numericCondition.even : numericCondition.odd;

        if (theNumber % 10 == 0) condition = numericCondition.multiple;
        if (theNumber == 0) condition = numericCondition.zero;
```

# Loop Øvelse

Loop øvelse:

- Lav et program der initialiserer en variabel kaldet i ved 0 og tæller opad fra 0. Initialiser en anden variabel kaldet j med værdien 25 og tæl nedad.

- Brug et enkelt for loop til at *inkrementere* i og *dekrementere* j simultant (samtidigt), og output værdierne for i og j ved hver iteration af dit loop.

- Når **i** er større end **j**, så afslut dit loop og print beskeden "Crossed over!"

# Loop Øvelse

Loop øvelse - hints:

- Studer syntaksen, der er tilladt i et standard for loop. Du vil være nødt til at deklarere i og j før, du starter et loop.
- I headeren skal du initialisere i ved 0 og j ved 25. Du vil gerne have loopet til at slutte når j er blevet mindre end i, så din betingelse er simpel: i < j.
- Du ønsker at i skal tælle opad og at j skal tælle nedad, så din iterator skal være **i++** og **j- -**.
- Så snart j bliver mindre end i, slutter dit loop, så du vil gerne kunne outputted den sidste besked efter afslutningen af dit loop.

# Greeter Robot Øvelse

Mål for øvelsen:

1) At lave en 'Greeter Bot' console app, som kan printe en hilsen ud til konsollen. Dette er kendt stof fra de foregående øvelser.

2) At lave ændringer i koden, sådan at din app ***returnerer*** input strengen til den kaldende metode, snarere end at den printer inputtet til konsollen.

3) Etabler et test project, som kan afvikle kode der checker at det robot objekt, vi ***instantiatierer*** vil returnere præcist den samme tekst, som vi sender ind.

# Greeter Robot Øvelse

Du kan starte ved at afvikle (Ctrl-F5) denne kode i en ny app:

```
using System;

namespace GreeterBot
{
    public class Program
    {
        static void Main(string[] args)
        {
            RobotVoice metalhead = new RobotVoice();
            metalhead.Say("Hey dude, how's it hangin'!");
        }

    }

    public class RobotVoice
    {
        public void Say(string passedStringArgument)
        {
            Console.WriteLine("Class RobotVoice says: " + passedStringArgument);
        }

    }
}
```

# Greeter Robot Øvelse

Mål nr. 2:
Now, make the robot return a string to the calling method.

Mål nr. 3 - hints:

Tilføj et 'test project' til din VS solution – GreeterBot.Tests. Husk at lave en reference i testprojektet til GreeterBot console app projektet.

Du kan nu ændre navnet og de metoder, der findes i testprojektets default klassefil, ***UnitTest1.*** Brug [TestClass] attributten til at ***decorate*** en ny klasse, 'public class BotTester'.

# Greeter Robot Øvelse

Øvelse 3 - hints fortsat:

Vi ønsker nu at teste, at vores program returnerer præcist det samme, som vi giver det som input. Vores metode:

```
[TestMethod]
public void SayHiReturnsHi()
{
```

.. og den vil gøre det følgende:

- lav et nyt  RobotVoice objekt.
- definer en string kaldet ‘statement’ som får værdien af et kald til RobotVoice objektets ‘Say’ metode, hvor vi passer "Hi” ind.
- Dette er den sidste linie: Assert.AreEqual("Hi", statement);

# Greeter Robot Øvelse

Til sidst skal du skrive denne metode ind i dit testprojekt. Prøv at ændre i den eksisterende kode så testen returnerer 'pass' i stedet for som nu 'fail'  Prøv først at beskrive for dig selv, hvilke ændringer vi skal lave for at få testen til at passe:

```
[TestMethod]
public void SayHello_ReturnsHelloWithRecipientName()
{
    RobotVoice metalhead = new RobotVoice("Finn");
    string statement = metalhead.SayHello("Santiago");
    Assert.AreEqual("Hello Santiago from Finn", statement);
}
```