

OOP 2020

Intro og Klasser

Thomas Bøgholm

Kursusintroduktion

- Kursus: Fredage 8:15-12:00
 - Forelæsning fra 10:15
- Spørgsmål?

Imperativ programmering

- Programmering via.
 - Funktioner
 - Kontrol-strukturer
 - Datatyper (int, float,..)
 - Komplexe datatyper
 - Structs
- Topdown programmering
 - Opsplitning af funktioner i mindre funktioner
 - Strukturering af program sker **efter funktionalitet**

JavaScript

- Programmering via.
 - Ovenstående + objekter
- Løsning af et specifikt problem
 - Funktioner løser en del af det større
- Rigtige programmer har ikke en enkelt top

Objekt-orienteret programmering

- Fokuserer bl.a. på modellering af begreber og genbrug
 - Programmer struktureres efter data
- Bottom-up programmering
 - Identifikation af mindre komponenter der kan sættes sammen til et større program
 - Programmer bygges som relationer mellem komponenter (objekter)

Objektorienteret Programming

- Objektorienteret programmering bygger på:
 - indkapsling
 - *abstraktion*
 - *nedarvning*
 - *Polymorfi*
- Nyttigt til at strukturere store systemer.

Resten foregår i visual studio

- Der er et antal slides inkluderet i dette slideset.
 - Det er slides fra tidligere år
 - De indeholder bl.a. det der er gennemgået via visual studio

IMPR eksempel

- Data
- Funktioner kaldt på data
 - Eller dele af
- Opdatering af data

```
typedef unsigned long int CPR;

struct person {
    char *name;
    CPR cpr, father_cpr, mother_cpr;
};

int main(void) {

    struct person baby, mother, father;

    baby.name = "Paul";
    baby.cpr = 2304031577;
    baby.father_cpr = 1605571233;
    baby.mother_cpr = 2412671234;

    mother.name = "Anna";
    mother.cpr = 2412671234;
    mother.father_cpr = 1605376789;
    mother.mother_cpr = 1201402468;

    father.name = "Frank";
    father.cpr = 1605571233;
    father.father_cpr = 1111331357;
    father.mother_cpr = 1212358642;

    printf("%s's mother and father is %s and %s\n",
        baby.name, mother.name, father.name);

    return 0;
}
```

IMPR eksempel (fra Kurts kursus)

- Data
- Funktioner
 - Mange funktioner
 - Spredt
 - Mange afhængigheder
- Opdatering af data
 - Mange opdateringer
 - Muligvis over alt

```
enum weekday {sunday, monday, tuesday, wednesday, thursday,
              friday, saturday};
typedef enum weekday weekday;

struct date {
    weekday day_of_week;
    int day;
    int month;
    int year;
};

typedef struct date date;

/* Is date d1 less than date d2 */
int date_before(date d1, date d2){
    return
        (d1.year < d2.year) ||
        (d1.year == d2.year && d1.month < d2.month) ||
        (d1.year == d2.year && d1.month == d2.month && d1.day < d2.day);
}

int main(void) {

    date today = {tuesday, 8, 11, 2011};
    date tomorrow = {wednesday, 9, 11, 2011};

    if (date_before(today, tomorrow))
        printf("OK\n");
    else printf("Problems\n");

    return 0;
}
```

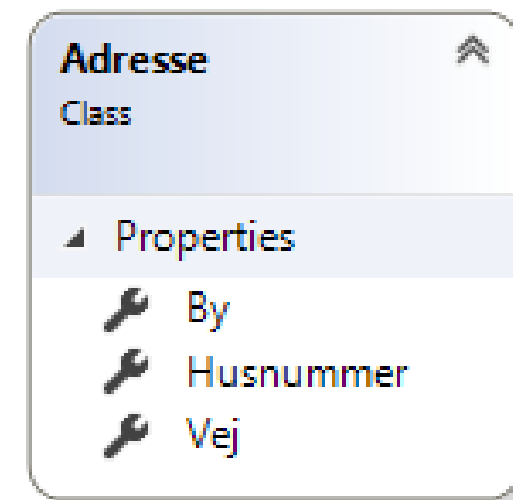
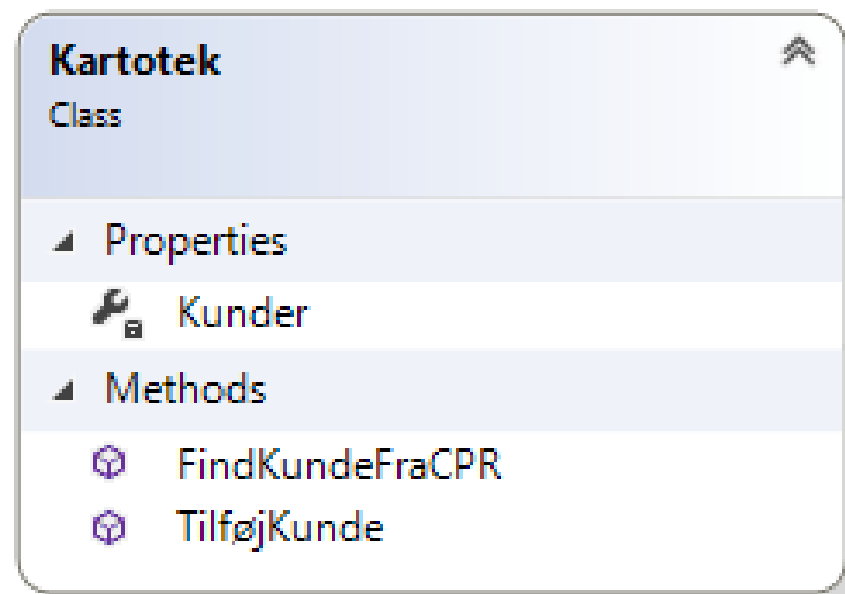
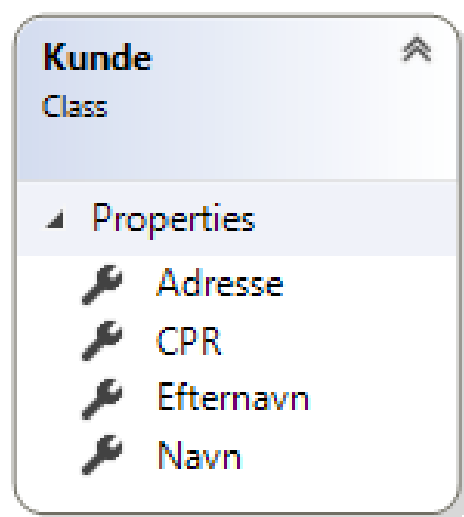

Klasser og objekter

- de mest centrale begreber i OOP!

Klassebegrebet

- En klasse er en *erklæring* af en samling af variabler og metoder (**type-erklæring**)
 - En opskrift på hvad objekter af *klassen* indeholder samt deres tilgængelighed
- Eksempel:
 - Klassen **Kunde**
 - Navn, Efternavn, [Adresse](#), CPR ...
 - Klassen **Kartotek**
 - KundeListe og metoder til at tilføje og søge efter kunder

Klassebegrebet



Klassebegrebet

```
class Kunde
{
    public string Navn { get; set; }
    public string Efternavn { get; set; }
    public Adresse Adresse { get; set; }
    public string CPR { get; set; }
}
```

```
class Adresse
{
    public int Husnummer { get; set; }
    public string Vej { get; set; }
    public string By { get; set; }
}
```

```
class Kartotek
{
    private List<Kunde> Kunder { get; set; }

    public void TilføjKunde(Kunde kunde)
    {
        Kunder.Add(kunde);
    }

    public Kunde FindKundeFraCPR(string CPR)
    {
        foreach (Kunde kunde in Kunder)
        {
            if (kunde.CPR.Equals(CPR))
                return kunde;
        }

        return null;
    }
}
```

Klassebegrebet

```
class Kunde
{
    public string Navn { get; set; }
    public string Efternavn { get; set; }
    public Adresse Adresse { get; set; }
    public string CPR { get; set; }
}
```

```
class Adresse
{
    public int Husnummer { get; set; }
    public string Vej { get; set; }
    public string By { get; set; }
}
```

```
class Kartotek
{
    private List<Kunde> Kunder { get; set; }

    public void TilføjKunde(Kunde kunde)
    {
        Kunder.Add(kunde);
    }

    public Kunde FindKundeFraCPR(string CPR)
    {
        foreach (Kunde kunde in Kunder)
        {
            if (kunde.CPR.Equals(kunde))
                return kunde;
        }

        return null;
    }
}
```

Klassebegrebet

```
class Kunde
{
    public string Navn { get; set; }
    public string Efternavn { get; set; }
    public Adresse Adresse { get; set; }
    public string CPR { get; set; }
}
```

```
class Adresse
{
    public int Husnummer { get; set; }
    public string Vej { get; set; }
    public string By { get; set; }
}
```

```
class Kartotek
{
    private List<Kunde> Kunder { get; set; }

    public void TilføjKunde(Kunde kunde)
    {
        Kunder.Add(kunde);
    }

    public Kunde FindKundeFraCPR(string CPR)
    {
        foreach (Kunde kunde in Kunder)
        {
            if (kunde.CPR.Equals(CPR))
                return kunde;
        }

        return null;
    }
}
```

Tænk på structs fra C

- bare med funktionalitet (metoder)

```
typedef struct trekant
{
    int h;
    int g;
} trekant_t;

...

int trekant_areal(trekant_t trekant)
{
    return trekant.h * trekant.g / 2;
}
```

```
class Trekant
{
    private int h;
    private int g;
    public int Areal()
    {
        return h * g / 2;
    }
}
```

Objektbegrebet

- Et *objekt* er en *indkapsling* af data
 - En sammenkobling af data og funktionalitet
 - “*bundling of data with the methods that operate on that data*”
- Et objekt har
 - En adresse i hukommelsen
 - En tilstand (variabler)
 - Fornavn, efternavn, adresse, alder, far, mor (sidste to: pointers til personer)
 - En opførsel (metoder – svarer til hvad i kender som funktioner)
 - Spis, drik, smil, vink

➤ Et *objekt* er en **instans** af en *klasse*

Klasser og Objekter

- **Person** er en klasse (begreb)
 - **Thomas** er en *instans* af person-klassen (konkret)
 - **Ole** er en *instans* af person-klassen (konkret)
 - **Band** er en klasse (begreb)
 - **Metallica** er en konkret *instans*
 - **Muse** er en konkret *instans*
- **Navn, bandmedlemmer, osv er forskellige!**
(og meget andet)

Objekter: data

```
typedef struct {  
    char *Fornavn;  
    char *Efternavn;  
    ...  
} Kunde;
```

```
Kunde *kunde = malloc(sizeof(Kunde));  
kunde->Navn = "Thomas";  
kunde->Efternavn="Bøgholm";  
...  
free(kunde);
```

```
class Kunde  
{  
    public string Navn { get; set; }  
    public string Efternavn { get; set; }  
    public Adresse Adresse { get; set; }  
    public string CPR { get; set; }  
}
```

```
Kunde kunde = new Kunde();  
kunde.Navn = "Thomas";  
kunde.Efternavn = "Bøgholm";  
...
```

Klassemedlemmer

- Felter → som medlemmer i CStructs
- Metoder → som funktioner i C, men altid defineret på klasser
- Properties → syntaktisk sukker for en sammensmeltning af metoder og felter
- Constructors → Initialisering

Klassemedlemmer: **felter**

```
class Kunde
```

```
{
```

```
    public string Navn;
```

```
    public string Efternavn;
```

```
    ...
```

```
}
```

Instansvariabler



Svarer til vores CStruct

```
typedef struct {  
    char *Fornavn;  
    char *Efternavn;  
    ...  
} Kunde;
```

Klassemedlemmer: Metoder

```
class Kunde
{
    public string Navn;
    public string Efternavn;
    public string FuldeNavn()
    {
        return Navn + " " + Efternavn;
    }
}
```


Pseudo-C-kode for metoden:



```
char* FuldeNavn(Kunde *kunde){
    int len=strlen(kunde->Fornavn) + strlen(kunde->Efternavn) + 2;
    char *result = malloc(len);
    strcpy(result, kunde->Fornavn);
    strcat(result, " ");
    strcat(result, kunde->Efternavn);
    return result;
}
```

Klassemedlemmer: Metoder

```
class Kunde
{
    public string Navn;
    public string Efternavn;
    public string FuldeNavn()
    {
        return Navn + " " + Efternavn;
    }
}
```



Scope: instans

Pseudo-C-kode for metoden:



```
char* FuldeNavn(Kunde *kunde){
    int len=strlen(kunde->Fornavn) + strlen(kunde->Efternavn) + 2;
    char *result = malloc(len);
    strcpy(result, kunde->Fornavn);
    strcat(result, " ");
    strcat(result, kunde->Efternavn);
    return result;
}
```

Klassemedlemmer: med modifiers

```
class Kunde
{
    public string Navn;
    public string Efternavn;
    private string CPR;

    public void SetCPR(string cpr)
    {
        if (IsCPRWelformed(cpr))
            CPR = cpr;
        else //Errorhandling
    }
    private bool IsCPRWelformed(string cpr)
    {
        ...
    }
    ...
}
```

Private access modifier

Kunde k = new Kunde();
k.CPR="asdf"; <- Ulovligt

CPR kan nu kun skrives med SetCPR
og hentes med GetCPR

Klassemedlemmer: Properties

```
class Kunde
{
    string Navn { get; set; }
    string Efternavn { get; set; }
    string CPR { get; set; }
}
```

Kort version

```
class Kunde
{
    private string _navn;
    private string _efternavn;
    private string _cpr;

    public string Navn
    {
        get { return _navn; }
        set { _navn = value; }
    }

    public string Efternavn
    {
        get { return _efternavn; }
        set { _efternavn = value; }
    }

    public string CPR
    {
        get { return _cpr; }
        set { _cpr = value; }
    }
}
```

```
class Kunde
{
    private string navn;

    public string GetNavn()
    {
        return navn;
    }

    public void SetNavn(string value)
    {
        navn = value;
    }

    private string efternavn;

    public string GetEfternavn()
    {
        return efternavn;
    }

    public void SetEfternavn(string value)
    {
        efternavn = value;
    }

    private string cpr;

    public string GetCPR()
    {
        return cpr;
    }

    public void SetCPR(string value)
    {
        cpr = value;
    }
}
```


Properties

- Properties er en speciel adgang til member variabler
 - Mulighed for validering ved get/set
 - read/write adgang (get/set)
 - Svarer til get/set-metoder fra andre sprog (f.eks. Java)
 - Get og set kan have forskellige access modifiers
 - **God praksis** at tilgå instans-variable via properties – også fra klassen selv! (validering & vedligeholdelse)
- **Automatiske properties** til ligetil tilgang (opretter skjult instans-variabel)
- **Virtuelle properties** (ingen tilsvarende instans-variabel)

```

class Person {
    //private instans-variable
    private string _navn; //kan overhovedet ikke ses udefra
    private double _vægt;
    private int _skoStørrelse;

    //properties der indkapsler instans- variable
    public int SkoStørrelse
    {
        get { return _skoStørrelse; }
        set { _skoStørrelse = value; }
    }
    public double Vægt
    {
        private get { return _vægt; }
        set { if (value > 40) _vægt = value; }
    }
    public string NavnOgVægt
    {
        get { return _navn + " vejer " + _vægt ; }
    }
    public int Alder { get; set; }
}

```

```
class Person {  
    //private instans-variable  
    private string _navn; //kan overhovedet ikke ses udefra  
    private double _vægt;  
    private int _skoStørrelse;
```

NB: '_' konvention.

//properties der indkapsler instans- variable

Typisk property

```
public int SkoStørrelse
```

synlighed for denne property

```
{
```

```
    get { return _skoStørrelse; }
```

```
    set { _skoStørrelse = value; }
```

keyword **value** = parameter til setter

```
}
```

```
public double Vægt
```

Write-only (udenfor klassen)
- Ændret (mere restriktiv) synlighed

```
{
```

```
    private get { return _vægt; }
```

```
    set { if (value > 40) _vægt = value; }
```

Validering

```
}
```

```
public string NavnOgVægt
```

Read-only (kun getter)

```
{
```

```
    get { return _navn + " vejer " + _vægt ; }
```

Udregnet property

```
}
```

```
public int Alder { get; set; }
```

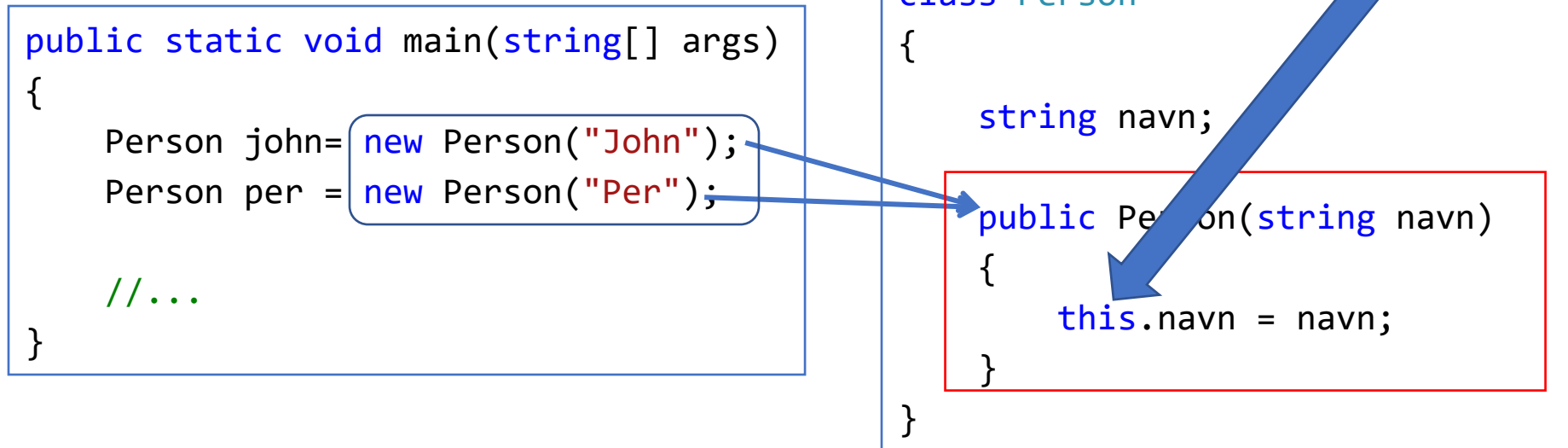
Automatisk property

```
}
```

Instantiering: Constructors

- **Constructor** er en speciel metode
 - kaldes når en instans skabes med **new** operatoren.
 - new allokerer plads i hukommelsen
- Constructoren til *initialisering*
- (**this** keyword bruges til eksplicit at angive instans-medlem)

this er en variabel der altid er tilgængelig i instans-scope (også metoder og properties) - peger på "nuværende instans"



Default constructor

- **Default constructor:** Tilføjes automatisk *med mindre* der eksplicit tilføjes en constructor.
 - Default constructor har *ingen parametre*

```
public static void main(string[] args)
{
    Person p1 = new Person("Ib");
    Person p2 = new Person();
}
```

Kompileringsfejl!

```
class Person
{
    public string navn;
    public double vægt;

    public Person(string navn)
    {
        this.navn = navn;
    }
}
```

Flere constructors

- Man kan bruge multiple constructors og **constructor chaining** til at undgå redundans.

```
class Person
{
    public Person(string navn)
        : this(navn, 50, 1.50)
    { }

    public Person(string navn, double vægt)
        : this(navn, vægt, 1.50)
    { }

    public Person(string navn, double vægt, double højde)
    {
        this.Navn = navn;
        this.Vægt = vægt;
        this.Højde = højde;
    }
}
```

Identificeres via navn OG parametre

Regel: Hvis redundans -> refaktorér

Objektorienteret Programming

- Objektorienteret programmering bygger på:
 - indkapsling
 - *abstraktion*
 - *nedarvning*
 - *Polymorfi*
- Nyttigt til at strukturere store systemer.

Indkapsling

- **Indkapsling:**

1. Gruppér data og metoder i én enhed
2. Skjule data og implementationsdetaljer for omgivelserne (information hiding).

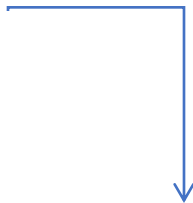
➤ Gør programmer nemmere at forstå og vedligeholde

➤ Ansvarsfordeling

Indkapsling på klasseniveau

- Klasser indkapsler **instans-variabler** og **-metoder** (medlemmer).
 - Instansvariabler har klasse-scope
- Instans-variable initialiseres til default-værdi
- **Instans-variable knytter sig til et objekt**

```
class Person {  
    //instans-variable  
    private string navn;    //null  
    private double vægt; //0  
  
    //instans-metode  
    public string NavnOgVægt() {  
        return this.navn + " vejer " + vægt + " kg.";  
    }  
}
```



Dårligt design

- Ofte rigtig dårlig ide med offentlige instans-variable!!
 - Ingen validering, ingen mulighed for at angive read-only tilgang eller skjule tilstand.

```
public static void main(String[] args)
{
    Person ida = new Person();

    ida.navn = "Ida Larsen",
    ida.vægt = -1;
    ida.telefonnummer = "112";
    ida.navn = "Børge Olsen";
}
```

• Fornuftig værdi?

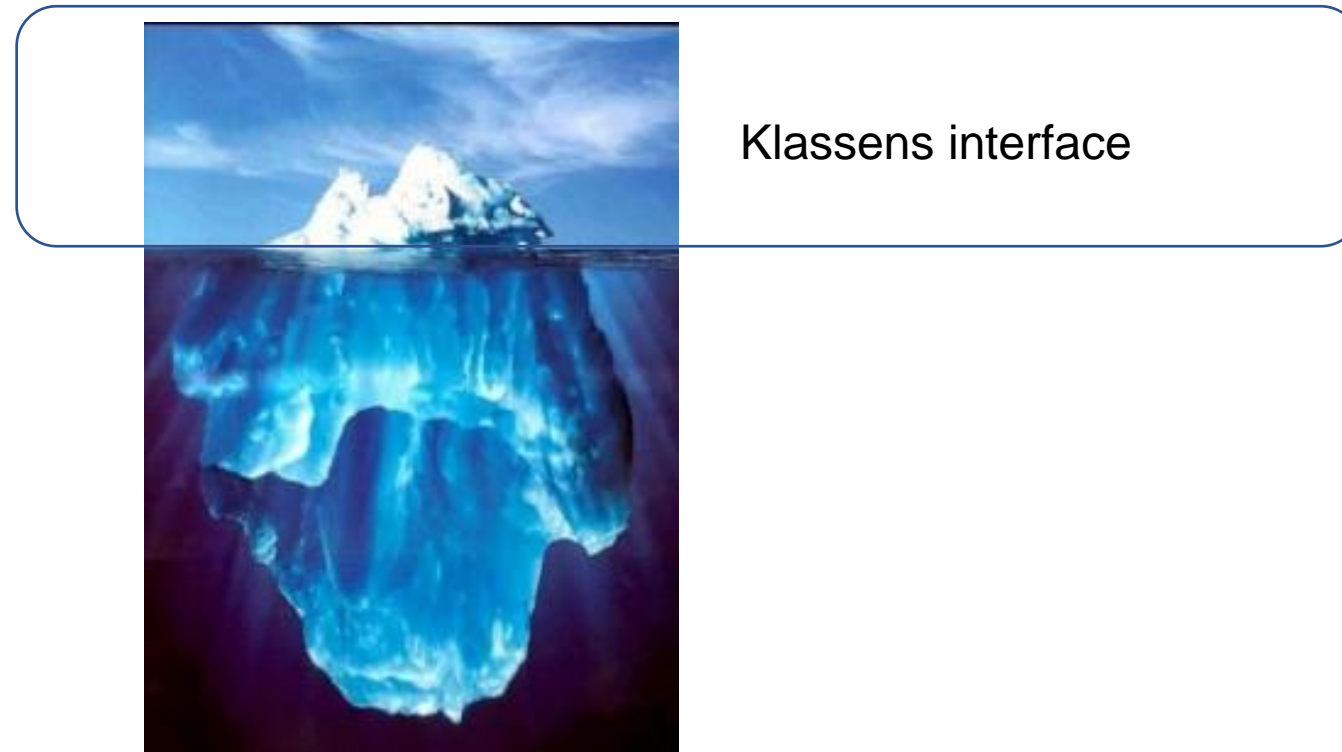
• Validering? Gem tidligere?

• Må ikke kunne ændres

•

Instans-metoder

- Hold det offentlige API minimalt!



- Ændringer i ikke-synlige metoder påvirker ikke klienter (vedligeholdelses-bonus)

OOP: Relationer mellem objekter

- Store systemer laves ved at danne relationer mellem objekter
- En relation dannes ved at tilføje en reference (som en instansvariabel) til en given klasse

```
class Kunde {  
    public string Navn { get; set; }  
    public Kunde(string navn) {  
        this.Navn = navn;  
    }  
}  
  
class Konto {  
    Kunde Kunde;  
    public int KontoNummer { get; set; }  
  
    public Konto(int kontoNummer, Kunde k) {  
        this.KontoNummer = kontoNummer;  
        this.Kunde = k;  
    }  
}
```

```
static void Main(string[] args)  
{  
    Kunde bo = new Kunde("Bo");  
    Konto ko = new Konto(123, bo);  
}
```

Kunne også sætte kunde
via property eller metode

Opsummering

- Klasser og objekter
 - **Klasse:** Skabelon hvorfra der skabes objekter
 - **Objekt** = **instans** af klassen.
 - Objekter **instantieres** med constructor.
 - Constructor foretager **initialisering** af **instans-variable**.
- En klasse **indkapsler** data og metoder
 - Bruger adgangs-modifikatorer til *information hiding*.
 - Adgang til instans-variable foregår gennem **properties**
- Store systemer laves ved at danne relationer mellem objekter.

do's and don'ts

- Aldrig public fields
 - Brug properties og gerne validering
 - Brug private, hvis du kan
 - Tilstand ændres med properties eller metoder
- Brug constructors
 - Objekter skal altid observeres i en meningsfyldt tilstand
- Små offentlige interfaces

Demonstration af Visual Studio

Ekstra slides: selvstudie med mindre der er tid

Programmeringssproget C#

- I dette kursus benytter vi C#
 - Udtales “C Sharp”
 - Ét objektorienteret sprog ud af mange
 - Bl.a. Java, C++, osv
 - Automatisk hukommelseshåndtering
 - Aldrig mere malloc/calloc/free (mere om det senere)
- en hurtig C til C# gennemgang

Fra C til C#

- C# har ligesom C :
 - Indbyggede simple typer
 - og en masse avancerede typer (klasser)
 - Arrays
 - Kontrol-strukturer
 - Iterative
 - Selektive
 - Funktioner (men her kendt som metoder)
 - Egne sammensatte typer
 - Primært “klasser”

Indbyggede typer

- Værdityper:
 - Numeriske typer:
 - Heltal og reelle tal (int,long / float,double,decimal)
 - Logisk type: bool
 - Tegn: char
- Referencetyper
 - string
 - Og mange mange flere

Indbyggede heltals-typer

Type	Size	Range (Inclusive)	BCL Name	Signed
sbyte	8 bits	-128 to 127	System.SByte	Yes
byte	8 bits	0 to 255	System.Byte	No
short	16 bits	-32,768 to 32,767	System.Int16	Yes
ushort	16 bits	0 to 65,535	System.UInt16	No
int	32 bits	-2,147,483,648 to 2,147,483,647	System.Int32	Yes
uint	32 bits	0 to 4,294,967,295	System.UInt32	No
long	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	System.Int64	Yes
ulong	64 bits	0 to 18,446,744,073,709,551,615	System.UInt64	No

Decimal notation:

Hexadecimal notation (**0x** præfix, 0-9, A-F):

int i = 127;

int k = 0x7F

Indbyggede floating point typer

Type	Size	Range (Inclusive)	BCL Name	Significant Digits
float	32 bits	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	System.Single	7
double	64 bits	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	System.Double	15-16

↑ Større tal, mindre præcision – videnskabelig beregninger

↓ Mindre tal, større præcision – finansielle beregninger

Type	Size	Range (Inclusive)	BCL Name	Significant Digits
decimal	128 bits	1.0×10^{-28} to approximately 7.9×10^{28}	System. Decimal	28-29

Decimal notation:

double billion1 = 1000000000

Exponential notation (E):

double billion2 = 1E9

Numerisk Suffiks

- Type-inferens: Hardcodede værdier (litteraler) fortolkes som `int` (derefter `uint`, `long`, `ulong`) eller `double` (afhængig af `.`)
- Suffiks kan tilføjes for at tvinge litteral til en bestemt data type:

Suffix Type	Numeric Suffix	Example
unsigned <code>int</code>	U or u	<code>uint x = 100U;</code>
<code>long</code>	L or l	<code>long x = 100L;</code>
unsigned <code>long</code>	UL or ul	<code>ulong x = 100UL;</code>
<code>float</code>	F or f	<code>float x = 100F;</code>
<code>double</code>	D or d	<code>double x = 100D;</code>
<code>decimal</code>	M or m	<code>decimal x = 100M;</code>

`float f = 6.5;`

`decimal d = 6.5;`

← Ikke accepteret af compiler – brug hhv. *f* og *d* suffiks

Numeriske konverteringer

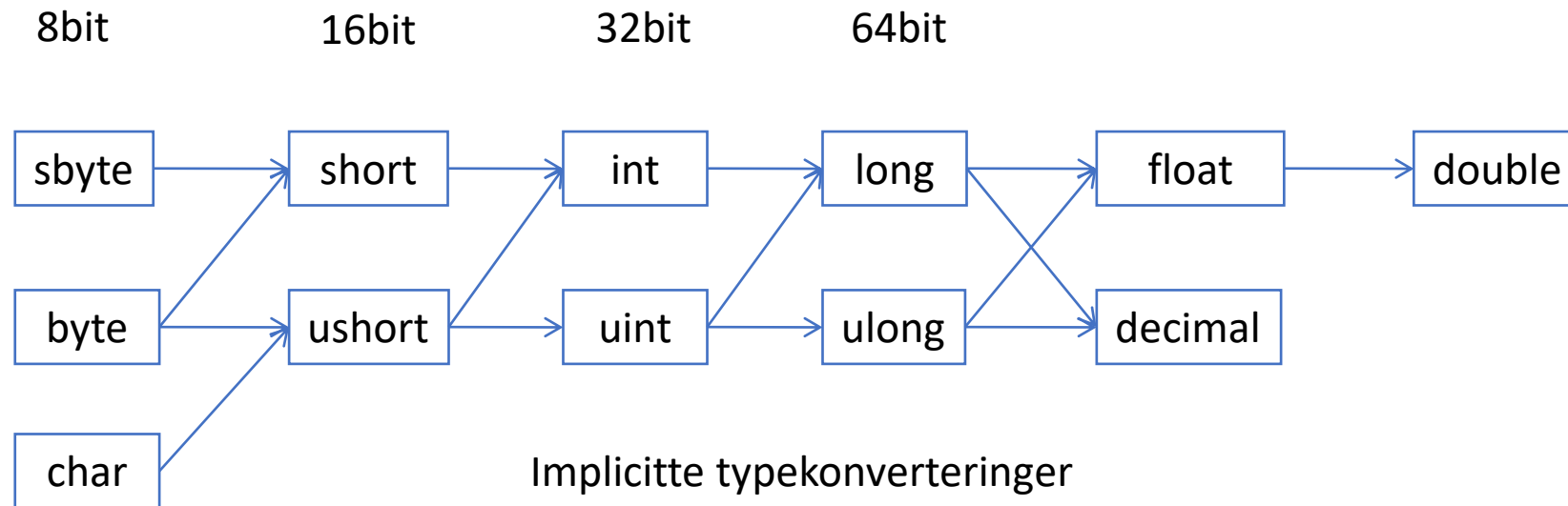
- Er **implicit** når *destinationstypen* kan holde alle værdier fra *sourcetypen*.
- Ellers kræves **eksplicit konvertering** (et *cast*).

```
short s1 = 10;
```

```
int i1 = s1;           //implicit typekonvertering
```

```
short s2 = (short)i1; //eksplicit typekonvertering (cast)
```

Implicitte typekonverteringer



Eksplisitte typekonverteringer

- Reelle tal -> heltal: Decimaler kappes af.

- `int i = (int)1.9f; // i er 1`

- `decimal` -> heltal

- `OverflowException`, hvis uden for range

- `double/float` -> heltal:

- Hvis værdi er uden for range:
 - Checked: `OverflowException`
 - Unchecked: Uspecificeret

```
double d = double.MaxValue;  
int x = checked ( (int) d );
```

- `Decimal` -> `float/double`:

- Afrunding til nærmeste værdi

- `float/double` -> `decimal`: Konvertering til decimal repræsentation

- Værdi for lille: 0.
 - for stort (absolut): `OverflowException`

bool og char typerne

- bool: true eller false.
 - Ingen konvertering til/fra andre typer
- char
 - Tegn (Unicode tegn – hvert tegn har unikt nummer)
 - Konverteres til ushort (og derfra videre)
 - <http://www.unicode.org/charts/>

- Tegnet 'X' på 4 forskellige måder:
 - `char x1 = 'X';` // literal
 - `char x2 = '\x0058';` // Hexadecimal
 - `char x3 = (char)88;` // Cast fra numerisk type
 - `char x4 = '\u0058';` // Unicode

Unicode Escape Characters

Escape Sequence	Character Name	Unicode Encoding
\'	Single quote	\u0027
\"	Double quote	\u0022
\\	Backslash	\u005C
\0	Null	\u0000
\a	Alert (system beep)	\u0007
\b	Backspace	\u0008
\f	Form feed	\u000C
\n	Newline	\u000A
\r	Carriage return	\u0009
\v	Vertical tab	\u000B
\uxxxx	Unicode character in hex	\u0029 (example)

String typen

- **string** klassen repræsenterer et array af char.
- String klassen har et rigt API bestående af e
 - **statiske metoder** (kaldes via klassenavn)
 - **instans-metoder** (kaldes via variabel-navn).

```
static void Main(string[] args)
{
    string s2 = "hello world";
    string s1 = new string(new char[] { 'H', 'e', 'l', 'l', 'o' });
    int isSame = string.Compare(s1.ToLower(), s2.ToLower());
}
```

Statisk metode

Instans-metode

Arrays

```
int[] integerArray1 = new int[10];  
integerArray1[0] = 0;  
integerArray1[1] = 1;
```

Arrays

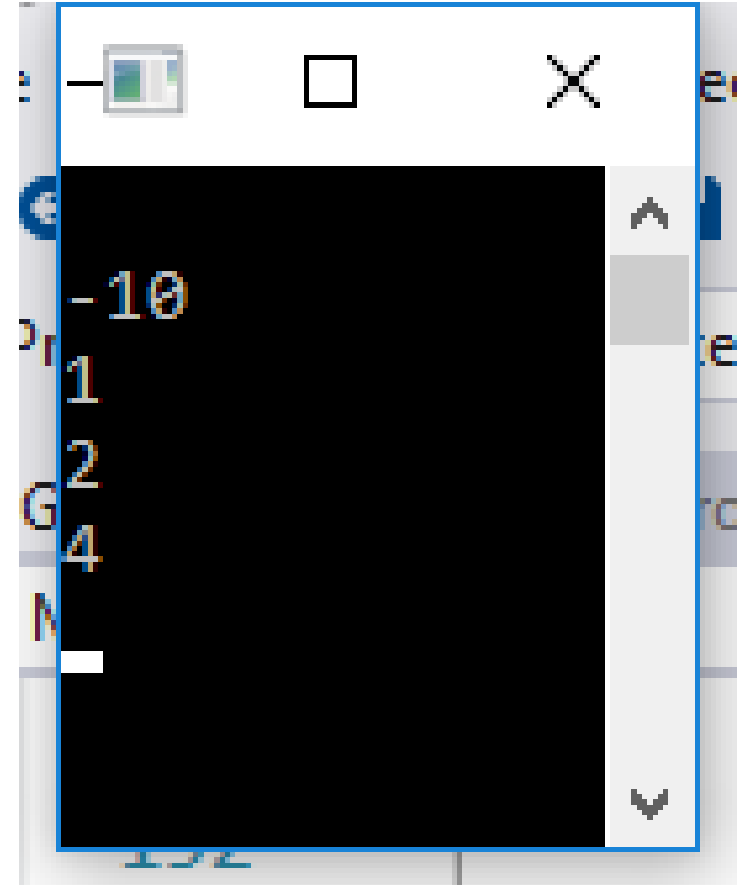
```
int[] integerArray1 = new int[10];  
integerArray1[0] = 0;  
integerArray1[1] = 1;
```

// kortere måder at lave arrays:

```
int[] integerArray2 = new int[] {1, 2, 3};  
int[] integerArray3 = new[] {4, 5};  
int[] integerArray4 = {6, 7, 8};
```

Lister

```
List<int> talListe = new  
List<int>();  
talListe.Add(1);  
talListe.Add(2);  
talListe.Add(3);  
talListe.Add(4);  
talListe.RemoveAt(2);  
talListe.Insert(0, -10);  
  
foreach (var tal in talListe)  
{  
    Console.WriteLine(tal);  
}
```



Kontrolstrukturer

- If-then else
 - Eksempel i VS
- Switch/case
 - Der kan switches på strenge
 - Eksempel i VS
- Betinget
 - `string s = ((1 == 2)? "ja" : "nej");`
- Løkker
 - Næste slides

Løkker (1)

- for, while, do/while
 - Skrives på samme måte som i C

```
int count = 10;  
for (int i = 0; i < count; i++)  
{  
    //kode  
}
```

Løkker (2)

- C# har også **foreach**

```
int[] intarray = new int[] {1,2,3,4,5};  
foreach (int i in intarray)  
{  
    // gør et eller andet fancy  
    // med værdierne 1 - 5  
}
```

Et simpelt Program

Vores hovedklasse

```
class Program
{
    static void Main(string[] args)
    {
        int[] intarray = new int[] {1,2,3,4,5};
        foreach (int i in intarray)
        {
            // gør et eller andet fancy
            // med værdierne 1 - 5
        }
    }
}
```

Main, som i C