

# OOP

---

**Uge 1:** Indkapsling, arv, abstraktion og polymorfi i  
klasseprogrammering

# Indhold

Målpind ( <b>Hvad</b> i skal lære og <b>hvorfor</b> i skal lære det) .....	3
• Teorier bag målpind .....	4 & 5
Plan for ugen ( <b>Hvor</b> og <b>hvornår</b> skal i lære det) .....	6
Ugens case ( <b>Hvordan</b> skal i lære det) .....	7 - 12

# Målpind

Undervisning indhold er indrettet for at opfylde følgende målpind stillet for faget.

1. Eleven kan anvende et objektorienteret programmeringssprog til at udarbejde konsolprogrammer, der indeholder flere klasser og er i overensstemmelse med OOP konceptet.
2. Eleven har en grundlæggende viden om det valgte programmeringssprog/framework.
3. Eleven kan definere og designe egne klasser.
4. Eleven kan erklære og instantiere objekter.
5. Eleven kan redegøre for typer af collections og kan udpege hensigtsmæssigt i forhold til et behov.
6. Eleven kan anvende en given kodestandard for det pågældende sprog.
7. Eleven kan håndtere "exception handling". 15-07-2017 og fremefter
8. Eleven kan redegøre for OOP konceptet såsom indkapsling, polymorfi og arv.
9. Eleven kan udarbejde en applikation som gør brug af OOP konceptet.
10. Eleven kan implementere abstrakte klasser og metoder.

Men **i denne uge**, dækkes kun de målpind som omhandler OOP og klasser, vises forneden i grønt:

1. Eleven kan anvende et objektorienteret programmeringssprog til at udarbejde konsolprogrammer, der indeholder flere klasser og er i overensstemmelse med OOP konceptet.
2. Eleven har en grundlæggende viden om det valgte programmeringssprog/framework.
3. Eleven kan definere og designe egne klasser.
4. Eleven kan erklære og instantiere objekter.
5. Eleven kan redegøre for typer af collections og kan udpege hensigtsmæssigt i forhold til et behov.
6. Eleven kan anvende en given kodestandard for det pågældende sprog.
7. Eleven kan håndtere "exception handling". 15-07-2017 og fremefter
8. Eleven kan redegøre for OOP konceptet såsom indkapsling, polymorfi og arv.
9. Eleven kan udarbejde en applikation som gør brug af OOP konceptet.
10. Eleven kan implementere abstrakte klasser og metoder.

# Teori bag målpind (og case)

## del 1 af 2, (*del 2 forsættes på næste slide*)

### Indkapsling/abstraktion

**Indkapsling** beskriver den handling, hvor man grupperer passende data sammen i passende type (klasse), og giver det type (klasse) en **abstrakt repræsentation** i form af et passende navn. Helst et passende navn som repræsenterer et objekt der eksisterer i virkeligheden.

- For eks. gruppering af data relateret til biler sammen i en klasse og give klassen navnet Car.cs.

Det samme gælder for kode som skal implementeres i en klasse. Koden skal grupperes f.eks. i en metode hvor metoden skal have en abstrakt repræsentation af denne kode samling i form af et passende navn.

- En kodestyk som f.eks. skal gemme bruger input data skal så f.eks. hedde SaveUserInput();
- Kun en funktion per indkapsling. Hvis SaveUserInput metoden også indeholder indsamling af bruger data og derefter gemmes, så skal indsamling af bruger data placeres i eget indkapsling, f.eks. GetUserInput(), som så efterfølgende kalder SaveUserInput().

Formål med indkapsling er:

- **Kode genbrug.** (Indkapslede kode i klasser kan instantieres fra andre klasser og metoder. Indkapslede kode i metoder kan kaldes fra andre metoder og objekter)

### Arv/abstraktion

**Arv** beskriver den handling, hvor flere relateret types (klasser) kan **indkapsles og abstrakt repræsenteret** som var de en enkelt type (klasse).

- Indkapsling af types i en arv er hierarkisk:
  - Base klasser, derived klasse og derived sibling klasser.
- Formål med arv er:
  - **Kode genbrug.** (Alle derived klasser kan bruge kode implementeret i base klasse)
  - At opnå abstraktion, hvor en klasse i højre hierarki (base klasse), kan dikterer hvilket members de klasser i laver hierarki (derived klasse) skal implementeres.

# Teori bag målpind (og case)

## del 2 af 2, (del 1 på forrige slide)

### Abstraktion med arv

- **Abstraktion i arv** beskriver den handling hvor en klasse i højre hierarki (base klasser), kan dikterer hvilket members de klasser i laver hierarki (derived klasser) skal implementeres.
  - Base klasser kan tilføje keyword: **abstrakt** til sin members (f.eks. en metode), som fritager disse members i at implementerer kode, men kræver så, at deres derived klasser skal implementer koden.
  - Hvis man i stedet anvender keyword **virtual**, i en base klasse members, f.eks en metode, skal metoden implementerer kode, men derived klasser må gerne overskriver koden med andet kode som passer bedst til derived klasserne.
  - Formål med abstraktion i arv:
    - **Kode genbrug.** (Genbruger metodens navn, mens metodens indhold er forskellige)
    - Separation af definition og implementation

### Polymorfi

- **Polymorfi** beskriver den handling hvor flere relateret metoder kan **indkapsles og abstrakt repræsenteret** som var de en enkelt metode, men med det egenskab at den kan håndterer forskellige datatype som parameter eller som retur værdi. .
- Eks: nedstående er 2 metoder indkapslet og abstrakt repræsenteret som en enkelt GetAge() metode der kan tage imod DateTime eller string som parameter:

```
int age = myClass.GetAge();  
▲ 1 of 2 ▼ int Enrollment.GetAge(DateTime date)
```

Mulighed 1 med DateTime som parameter

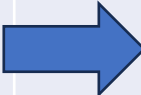
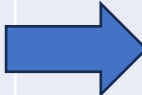
```
int age = myClass.GetAge();  
▲ 2 of 2 ▼ int Enrollment.GetAge(string date)
```

Mulighed 2 med string som parameter

- Formål med abstraktion i arv:
  - **Kode genbrug** (men i objekt niveau).

# Plan for ugen : første uge

Denne uge skal vi bruge til at dække alle målpind der har med OOP at gøre.

Dag 6	Dag 7	Dag 8	Dag 9	Dag 10
Intro til faget	Opsamling	Opsamling	Opsamling	Opsamling
<p>Vi skal med code-along og ude fra en case, opleve hvordan type systemet hænger sammen med OOP:</p> <ul style="list-style-type: none"><li>• Indkapsling af data, og types</li><li>• Abstraktion i indkapsling</li><li>• Indkapsling af types med arv.<ul style="list-style-type: none"><li>• Base klasse constructor</li><li>• Arv keywords</li></ul></li></ul>	 Forsat fra i går.	<p>Vi skal med code-along udbygge vores case, for at opleve hvordan arv understøtter:</p> <ul style="list-style-type: none"><li>• Abstraktion</li></ul> <p>Og hvordan types understøtter:</p> <ul style="list-style-type: none"><li>• polymorfi.</li></ul>	 Forsat fra i går.	<p>Vi skal med code-along udefra en case, opleve hvordan OOP koncept går begge vej hvor vi kan starte med abstraktion for at opnå</p> <ul style="list-style-type: none"><li>• Indkapsling og</li><li>• arv</li></ul>

# Code-along case

---

Case: Udvikling af en C#/.NET konsol applikation til TEC, som TEC kan bruges til at tilmelde elever til fagene på H1.

- I denne casen skal eleverne ikke udvikle applikation individuelt, men som **code-along øvelse i 4 dele** sammen med læreren.
  - Når alle 4 dele er gennemført, vil applikation være færdig implementeret.

*Dog vil der være øvelse som overlapper grundlæggende klasse programmering med grundlæggende kontrolstruktur osv, som elev selv skal implementerer færdig selvstændigt.*

# Øvelse 1/4 - Indkapsling

---

## 1.1 Vi Opretter en konsol app med navn: TecFagTilmeldingApp

- Applikation skal udvikles for TEC, og skal bruges til at tilmelde elever til fagene på TEC.

## 1.2 Det er afgjort, at nedstående data skal indgå i applikationen:

- Elev: for- og efternavn, fødselsdato og alder.
- Lære: for- og efternavn , fødselsdato, alder og afdeling.
- Fag: navn og lærer.

**Indkapsle data foroven i passende klasser** (*klasse navne skal passende repræsenterer de data indkapslet i klassen*).

**Formål: kode genbrug.** (Når data er indkapslet i klasser, kan ALLE andre klasser og/eller metoder instantierer klassen og bruge det.)

**Bemærk:** Alder skal **IKKE** sendes med til 'constructor', i skal implementerer en funktionalitet som skal beregn alder ude fra fødselsdato!

**Forsættes på næste slide....**



# Øvelse 1/4 - Indkapsling

---

**Forsættes fra sidste slide....**

**1.3** Vi skal gennemgå de oprettede types(klasser) og se, om der er funktionaliteter(kode logik) i nuværende indkapsling, som eventuelt kan trækkes ud og indkapslet for sig selv i en separat klasse, eller metode.

- **Formål: kode genbrug.**

*( Der er jo den funktionalitet som konverterer fødselsdato til alder. Kan andre i nuværende eller fremtidig klasser genbrug dette? Ja, den kan indkapslet i eget klasse og genbruges af Elev og Lærer klasser fordi de begge har alder property. )*

**1.4** Når vi nu mener at alt vores data er indkapslet, skal vi overveje, om nogle af vores klasser skal sendes vider til andre klasser.

- Ja, det er der: Fag klassen skal indeholde fagets navn, men også få tilsendt en lærer objekt (underviser af faget).  
Derfor skal vi nu overveje, om der er behov for at sende hele lærer objekt (med fields, property, constructor og metoder), eller om det er nok, kun at sende lærerens for- og efternavn afsted (lærerens for-og efternavn property).

Fordi det er nok kun at sende for- og efternavn afsted, skal man overveje at indkapsle for- og efternavn for sig selv (i en model klasse). **Formål: kode genbrug.**

**Forsættes på næste slide....**

# Øvelse 2/4 – arv

Arv er ikke en selvstændig del i OOP! Arv er en indkapsling teknik. Klasser kan være relateret og høre sammen. Anvend arv som indkapsling teknik, til at indkapsle flere samhørende klasser sammen i en struktur: en arv struktur.

## Forsættes fra sidste slide....

**2.1** Når data er indkapslet i passende klasser, skal man tjekke om der er 'redundant' (overflødig) members (property og metoder) i klasserne. Er der det, skal man overveje **arv** for at **genbrug** de gentaget members.

- Student og Teacher klasser har for- efternavn, fødselsdato og alder som overlapper hinanden. En base klasse som Student og Teacher kan arve fra (Person.cs) kan løse problemet for de gentaget property, samt de gentaget initiering i constructor!*

```
public PersonModel? PersonalInfo { get; set; }  
1 reference  
public DateTime BirthDate { get; set; }  
1 reference  
public int Age { get; set; }  
3 references  
public string? Department { get; set; }  
4 references  
public Teacher(string? firstName, string? lastName, DateTime birthDate, string? department)  
{  
    PersonalInfo = new() { FirstName = firstName, LastName = lastName };  
    BirthDate = birthDate;  
    Age = new AgeConverter(birthDate).Age;  
    Department = department;  
}
```

```
public PersonModel? PersonalInfo { get; set; }  
1 reference  
public DateTime BirthDate { get; set; }  
2 references  
public int Age { get; set; }  
1 reference  
public Student(string? firstName, string? lastName, DateTime birthDate)  
{  
    PersonalInfo = new() { FirstName = firstName, LastName = lastName };  
    BirthDate = birthDate;  
    Age = new AgeConverter(birthDate).Age;  
}
```

Ens properties og ens initiering, **arv** kan reducerer dette til 1 enkelt.

Forsættes på næste slide....

# Øvelse 2/4 – arv

---

Forsættes fra sidste slide....

**2.2 Hvordan læser vi nu alle de data fra vores klasser? Lad os prøve at lave en test læsning: prøve i Program.cs at opret et fag, en lærer og en elev.**

```
Teacher niels = new("Niels", "Olesen", new DateTime(1971, 2, 23), "CIT");  
Student patrik = new("Patrik", "Nielsen", new DateTime(1996, 8, 1));  
Course course = new("OOP", niels);
```

Og med Console.WriteLine(), udskrive følgende:

```
Niels alder: 52  
Patriks alder: 27  
OOP lærer er: Niels Olesen
```

Forsættes på næste slide....

# Øvelse 2/4 – arv

## Forsættes fra forrige slide

### 2.5 Ude fra følgende data :

- Niels Olesen : Grundlæggende programmering, OOP, Studieteknik
- Henrik Paulsen : Netværk
- Jack Baltzer : Clientside programmering, Database programmering
- Bo Elbæk : Computerteknologi

Opret alle lærer og derefter fag som objekter (give alle lærer 'CIT' som afdeling, opdigt selv lærerens fødselsdato).

### 2.6 Med det vi indtil nu har implementeret, færdiggøre applikation, som skal fungerer som følgende:

```
Ingen elever er tilmeldt endnu  
Angiv elev fornavn:
```

1. Når app startes

```
Ingen elever er tilmeldt endnu  
Angiv elev fornavn: Jens  
Angiv elev efternavn: Jensen  
Angiv elev fødselsdato: 23.02.1998  
  
Fag id: 1, fag navn: Grundlæggendeprogrammering  
Fag id: 2, fag navn: OOP  
Fag id: 3, fag navn: Studieteknik  
Fag id: 4, fag navn: Netværk  
Fag id: 5, fag navn: Clientside programmering  
Fag id: 6, fag navn: Database programmering  
Fag id: 7, fag navn: Computerteknologi  
Angiv ID for det fag som elev skal tilmeldes: 2  
  
Jens Jensen er nu tilmeldt OOP  
Tilmeld ny elev [J/N]:
```

2. App spørger om elevens for-, efternavn, fødselsdato, samt det fag (ude fra en list af alle fag) som eleven skal tilmeldes.  
App bekræfter herefter at eleven er tilmeldt.  
App spørger om der skal oprettes en ny elev.

```
Tilmeldte elever:  
Jens Jensen (25 år gammel) er tilmeldt OOP  
Angiv elev fornavn: _
```

3. Hvis "j" vælges, clear skærm, vis tidligere tilmeldte elever MED ALDER og spørge igen om info til den næste elev.

# Øvelse 3/4 – Abstraktion i arv

---

## Abstrakt/override

(Samme metode, men tvunget forskellige kode implementation.)

### 3.1 Give både elev og lærer klasse følgende fælles metode:

**'GetInfo(List<Enrollment> enrollments)'** :

- Når en **elev** objekt kalder metoden, skal metoden returnerer alle fag som eleven er tilmeldt.
- Når en **lærere** objekt kalder metoden, skal metoden returnerer de fag han har elever tilmeldt (det er ikke sukkeret, at der er nogen elever tilmeldt nogle af hans fag).

## Virtual/override

(Samme metode, med samme implementation. Men man kan ændre det original kode implementation med noget andet hvis man vil.)

### 3.2 Give både elev, lærer og deres base klasse en fælles metode **'ShowHolydays()'** :

- Når en **elev** objekt kalder metoden, skal metoden returnerer teksten "6 uger sommerferie, og 2 uger juleferie."
- Når en **lærere** objekt kalder metoden, skal metoden returnerer teksten "3 uger sommerferie og 3 uger selvvalgt ferie."
- Hvis elev eller lærer klasser **IKKE** vælger at implementerer metoden, skal metoden stadig kunne kaldes (fra base klassen) og returnerer teksten "Ferie ikke angivet."

# Øvelse 4/4 – Polymorfi

---

**4.1** Applikation skal implementerer følgende CountryCode **enum**: DK og EN.

**4.2 Både elev og lærer** skal kunne kalde følgende metode: **ShowBirthDate**.

- Man skal kunne kalde metoden uden parameter: **ShowBirthDate()**, Eller med CountryCode enum som parameter: **ShowBirthDate(CountryCode brugerDefineretFormat)**
  - Hvis metoden kaldes uden parameter:
    - Kaldes (enten fra elev eller lærer klasse), skal den returnerer elev eller lærerens fødselsdato som 'default' i engelsk format (year.month.day).
  - Hvis metoden kaldes med CountryCode enum som parameter :
    - Kaldes (enten fra elev eller lærer klasse), skal den returnerer elev eller lærerens fødselsdato i den format som enum dikterer year.month.day(EN) eller day-month-year(DK).