OOP

Uge 2: Kodestandarder, exception håndtering.

Der udleveres bedømmelse opgave (case)

Indhold

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 kodestandard og exception håndtering, 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

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

Kodestandarder

En kodestandard er afgørende for:

- St opretholde kodelæsbarhed, konsistens
- Et grundlag i samarbejde inden for et udviklingsteam.
- At følge industripraksis og etablerede retningslinjer hjælper med at sikre, at kode er nemmere at forstå, vedligeholde og udvide.
- At håndhæve en ensartet stil gennem kodekonventioner.
- **Fordi .NET tidligere har været forbehold Windows, men sener hænd gået cross-platform, så er det en standard, at udvikleren tester deres kode på alle platform. Bruger man Windows OS, er Linux test miljø indbygget i Visual Studio (både IDE og Code).**

Alt konventioner fra Microsoft:

https://learn.microsoft.com/en-us/dotnet/standard/design-guidelines/

Gode at kende fra begyndelse:

https://www.dofactory.com/csharp-coding-standards

Exception håndtering:

Fejl i koden kan forbygges. Der er indbygget kode skrivning teknikker som fortrækkes til at forbygge fejl end at indsæt alt i try/catch.

Exception klassen giver også mulighed for at implementer en tvang forsat eksekvering af sin kode når/hvis en exception forkommer.

Man kan med Exception klassen definer hvad man selv ønsker at programmet skal betragtes som fejl. Og derved kan man hefter fange sin eget definition af fejl i sin catch blok.

Plan for ugen : anden uge

Dag 6	Dag 7	Dag 8	Dag 9	Dag 10
Opsamling	Opsamling	Opsamling	Opsamling	Opsamling
• Exception håndtering • Udlever/gennemgå af bedømmelse case, samt bedømmelsestider til på fredag.	Vi skal se specifikt på følgende målpind: Eleven kan anvende en given kodestandard for det pågældende sprog. • Kodestandarder fra Microsoft for C#. • Cross-platform kodestandarder: Meget kode fra .NET Framework tiden virker stadig fint, men er ikke cross-platform! Hvordan kan i sikker jer, at jeres app bruger kun de kode til cross-platform.	Selvstændig arbejdsdag Arbejde vider på case 1. Både Lokale E311 samt Zoom link (for dem der arbejder ekstern) vil være åben i tidsperioden 8 – 14:15.	Selvstændig arbejdsdag Færdiggørelse **og aflevering** af case. Både Lokale E311 samt Zoom link (for dem der arbejder ekstern) vil være åben i tidsperioden 8 – 14:15.	Bedømmelse ude fra de tider i har fået.

Øvelse 1, forebyggelse del 1 af 2

se del 2 på næste slide

Teori:

Der er en udbredt fremgangsmåde indenfor programmering, at man skal kode med henblik på at forebygge fejl, og at dette bør praktiseres som en del af exception håndtering. Eks:

- Overveje hvorfor der f.eks. findes `Convert.ToDateTime(…)`, og sener hende blev der tilføjet `DateTime.TryParse(…)` til værktøjet kassen.

En af konverteringsteknik beskrevet foroven SKAL 'exception håndteres', mens den anden ikke skal, men dog alligevel er håndtering af mulig exception behandlet.

Øvelse:

I en konsol app, spørge bruger om at indtaste sin fødselsdato.

Implementerer både Convert og TryParse teknikken og konverter fødselsdato fra string til DateTime.

Begge skal udskrive "Format af din input er forkert!" hvis brugeren f.eks. Indtaster "aaa".

Tips: Den ene teknik skal bruge

```
try
{
   …
}
catch(…)
{
}
```

Den anden behøves ikke try/catch, men man kan fange fejlen alligevel og kan bruge følgende til at udskrive fejlen:

```
if(…) { … }
else { … }
```

Øvelse 1, forebyggelse del 2 af 2

se del 1 på forrige slide

```
// Kode eksempel 1:
string myTextFile = "MyTextFile.txt";
IEnumerable<string> lines = File.ReadLines(myTextFile);
```

```
// Kode eksempel 2:
string? myTextFile = "MyTextFile.txt";
StreamReader lines = File.OpenText(myTextFile);
string? line;
while ((line = lines.ReadLine()) != null)
    Console.WriteLine(line);
```

```
// Kode eksempel 3:
string myTextFile = "MyTextFile.txt";
if(File.Exists(myTextFile))
{
    using (StreamReader lines = new StreamReader(myTextFile))
    {
        string line;
        while ((line = lines.ReadLine()) != null)
            Console.WriteLine(line);
    }
}
```

Øvelse:

Til venstre er 3 kode eksempler som gøre præcis det samme ting. Hvilket 1 af de 3 har fejl forbyggelse, og hvilket KRÆVER som minimun, at være placeret i en try/catch blokke?

- Implementerer de 3 kode eksempler og se hvilket vil ”crash” hvis filen ikke findes.
- Med Microsoft learn link i har fået, gå til følgende link Og skrive som kode kommentar over det 3. eksempel hvad using gøre.

  https://learn.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=net-7.0

Øvelse 2, try, catch, finally

Teori:

try/catch har en svaghed i, at den tillader at systemet hopper ud af try blokken når fejlen opstår **fra og med det linje fejlen opstår** for at hoppe ind i catch blokken og derved få fejlen håndteret.

Nogle gang er der 'dependency' (afhængigheder) i det allerede startet kode stykke inden den afbrydes pga fejlen og hopper in i catch blokken.

Man kan derfor knytte endnu en blokke (finally) til try/catch.

Enhver kode i en try blokke med finally blokke tilknyttet, vil køre igennem finally blokke ligegyldig om koden køre fejlfri eller crasher. I finally blokke kan man så implementerer håndtering af disse 'dependency'.

Øvelse:

Følgende metode skal når den kaldes, holde styr på hvor mange gange den blev kaldt med property GetAgeAccessCout. men GetAgeAccessCout++ blev alrig kaldt fordi både try og catch har en return i sig, men hvis man indsætter GetAgeAccessCount++ inde i både try og catch, så bryder man OOP regel med at der ikke skal være kode gentagelser. Hvordan kan det så løses?

1. Opret en klasse i konsol app og indsæt property og metoden i klassen.
2. Kald metoden fra Program.cs og udskrive:
    1. Alderen og antal gange metoden kaldes.
    2. Fejlen hvis catch blokken køres og antal gange metoden kaldes.

```
public static int GetAgeAccessCount { get; set; }
public static string GetAge(string fødselsdato)
{
    int age = 0;

    try
    {
        //GetAgeAccessCount++;
        age = DateTime.Now.Year - Convert.ToDateTime(fødselsdato).Year;
        return $"Din alder er {age}";
    }
    catch (Exception exc)
    {
        //GetAgeAccessCount++;
        return exc.Message;
    }

    GetAgeAccessCount++;
}
```

Øvelse 3, throw

Teori:

Der vil i nogle situationer være behov for, at udvikleren selv at setter kondition for, at noget skal betragtes som fejl og udløser en exception, selv om koden køre rigtigt.

Med 'thorw' exception, kan vi selv bestemme hvad skal være fejl og udløser advarsler til brugeren.

En tomt list<Person> efter et database kaldt vil ikke udløse en fejl. Men udvikleren kan sidde med en insider viden om, at list af personer efter database kaldet ikke bør være tomt. Udvikleren kan derved set kondition for, at en tomt list skal udløse en exception:

```
List<Person> employees = GetAllEmployees();
If(employees.Count == 0)
    throw new Exception(”Person list er tomt!”);
```

Øvelse: Med følgende kode logik, lav en if-kondition som udløser en crash, hvis alderen er over 50 år.

1. Exception skal have følgende 'message' : ”You should be retired!”.
2. I skal herefter ”fange” denne crash og udskrive dens exception message til konsolen.
   - **Crashen skal fanges, DET MÅ IKKE VÆRE:**

```
if(age > 50)
{
   Console.WriteLine(”You should be retired!”);
}
```

```
Console.Write("\nAngive fødselsdato: ");
string fødselsdato = Console.ReadLine();

DateTime dato;
bool isDateTime = DateTime.TryParse(fødselsdato, out dato);
if(isDateTime)
{
    int age = DateTime.Now.Year - dato.Year;
}
```