Dag 3 Øvelser

Øvelse 3.0

a.

Lav (som i slides) en klasse Shape med attributter x og y (double) og tilhørende properties og passende constructors. Lav en constructor, der initialiserer x og y, og en parameterløs contructor. Den parameterløse skal initialisere til 1,1 og gøre dette ved at kalde den anden constructor.

b.

Lav underklasser til Shape:

- Circle, med yderligere property Radius (double), tilret constructor, så den også tager radius med (og kalder Shape's constructor)
- Rectangle, med yderligere properties Length og Width (double) og tilret ctor.

Test i en main-metode. Lav en liste af Shape-objekter, tilføj nogle til denne og gennemløb den med en løkke og udskriv oplysninger om hvert objekt.

c.

Lav en abstrakt metode public double Area() i Shape.

Lav implementationer af denne i underklasserne.

Udskriv Area i løkken fra før.

Øvelse 3.1

Skitser fire klasser i en generaliseringsstruktur, der repræsenterer begreberne *medarbejder*, *mekaniker*, *værkfører* og *synsmand* (på et mekanikerværksted).

Så tegn et diagram over det nedenstående og hvad der skal være med i de forskellige klasser – først i opgave 2 skal du i gang med at kode!

For *medarbejdere* registreres navn og adresse. Det skal være muligt både at opdatere og hente navn og adresse (altså via en property – du kan jo bruge auto-implemented properties, eller du kan bruge private fields og så public property metoder)

For *mekanikere* registreres endvidere et årstal for svendeprøve samt en timeløn. Igen skal det være muligt at hente og opdatere begge attributter. Mekaniker skal nedarve fra Medarbejder klassen.

For *værkførere* (der også er *mekanikere, så de nedarver fra Mekaniker klasse*) registreres år for udnævnelse til værkfører samt størrelsen af det tillæg pr. uge, som værkføreren gives ud over den almindelige timeløn.

En *synsmand* er også ansat på et værksted og er *mekaniker*. For hver Synsmand holdes rede på, hvor mange syn han har lavet på en uge. Hans ugeløn beregnes som antal syn i den pågældende uge * 290.

Øvelse 3.2

Opret en solution som en konsolapplikation (File, New, Project, Visual C#, Console Application). Giv den et fornuftigt navn og omdøb også Program.cs (højreklik på filen i Solution Explorer).

Programmér de fire klasser fra øvelse 1 og deres indbyrdes sammenhæng. For hver klasse: Opret en ny klasse i projektet (Project, Add Class og giv den navn). Så en klasse per fil!!!

Test, at der kan oprettes instanser af klasserne og indsættes og læses oplysninger fra din main metode.

Øvelse 3.3

Udvid klasserne fra opgave 1 og 2, så det nu bliver muligt at beregne ugeløn for alle. Det kan antages, at alle har en arbejdsuge på 37 timer.

Så du kan f.eks. tilføje følgende i Medarbejder klassen:

```
private const int _timerPrUge = 37;

        public virtual double BeregnUgeLøn()
        {
            return 0.0;

        }

        public int TimerPrUge  //og den tilhørende access metode, bemærk readOnly, dvs get!
        {
            get { return _timerPrUge; }

        }
```

Du skal så override BeregnUgeLøn() i underklasser og beregne den korrekte ugeløn for de forskellige typer af personale.

Det vil måske være fornuftigt at lave BeregnUgeLøn (og Medarbejder) abstract.

Test den nye funktionalitet.

Øvelse 3.4

Tilføj en CprNr klasse – der vil constructoren i den klasse se således ud:

```
  public CprNr(String bDate, String sNumber)
        {
            BirthDate = bDate;
            SequenceNumber = sNumber;

        }
```

Og her skal du så også definere to properties (BirthDate og SequenceNumber) med de korrekte typer i CprNr klassen – du kan lave dem som autoimplemented properties.

Tilret Medarbejder klassen fra foregående, den skal tilføjes et cpr nummer, dens constructor får nok denne signatur: (så du skal tilføje en property til klassen, således at data fra constructoren kan gemmes selvfølgelig).

```
public Medarbejder(CprNr cpr, String navn, String adresse)
```

Gør det samme med en String medarbejdernummer.

Override ToString metoden i Medarbejder så du kan udskrive medarbejder objekter til consolen – output for en medarbejder bestemmer du selv hvordan skal se ud.

Definerer et par Medarbejder objekter og udskriv dem til konsollen. Hvis du gerne vil formatere tingene til f.eks. blot at være på en linje, så kan nedenstående links måske hjælpe dig.

x: https://stackoverflow.com/questions/2978311/format-a-string-into-columns (Links til en ekstern webside.)
Or: (Too long) https://docs.microsoft.com/en-us/dotnet/api/system.string.format?view=netframework-4.7.2 (Links til en ekstern webside.)

Øvelse 3.5

I samme consol program som i øvelse 1-4, implementer (ved hjælp af generics – se .pdf fil om det på canvas) en selv-defineret collection klasse, som så kan indeholde Medarbejdere. Din collection klasse skal have de følgende metoder:

- void AddElement(TKey k, Medarbejder p)
  Inserts a Medarbejder p into the collection. The function must fail if the key is already occupied by some other element.
- Medarbejder GetElement(TKey k)
  Retrieves a Medarbejder identified by the TKey k. If not found then the function returns null.
- Int Size()
  Returns the number of elements in the collection.

Du kan bruge dictionary, som en intern variabel til at gemme dine ting i collection, så definition vil se således ud:

```
class MedarbejderListe<TKey >
    {

        private readonly Dictionary<TKey, Medarbejder > _collection = new Dictionary<TKey,
Medarbejder >();
```

Du skal lige have en Using på for at kunne bruge Dictionary:

```
 using System.Collections.Generic;
```

Der er ikke behov for en constructor – der vil som standard jo altid være en tom constructor.

Det er så meningen at du skal kunne gemme medarbejdere (eller instanser af underklasser af Medarbejder) i denne collection og så bruge Cpr-nummeret fra den foregående opgave. Det kunne se således ud:

```
static void Main()
        {
             var medarbejderCollection = new MedarbejderListe<CprNr>();

             var morten = new Mekaniker("Morten", "Brabrand", new CprNr("211271",
             "7777"), "1320", new DateTime(2017, 11, 23), 195);
             var karina = new Mekaniker("Karina", "Aarhus", new CprNr("141174",
             "8888"), "1410", new DateTime(2019, 1, 20), 190);

             medarbejderCollection.AddElement(karina.Cprnr, karina);
             medarbejderCollection.AddElement(morten.Cprnr, morten);
```

Test også at din Size() og din GetElement metode virker efter hensigten med noget kode i din main metode. F.eks. efter at have tilføjet to elementer, skal Size() jo returnere 2. Hvis GetElement ikke virker, så læs de næste sætninger.

Gad vide, om det er nødvendigt at lave en Equals metode i CprNr?. Undersøg, om dette er nødvendigt.

Måske virker det heller ikke med en Equals metode? Lav også en GetHashCode metode. Reglen er, at hvis to objekter er Equal skal de returnere samme hashcode.

Lav derefter, så Key er en String, der angiver medarbejdernummer. Tilret, lav en ny Main eller lav en ny metode, der kaldes fra Main.

Hint: du skal ikke lave om i MedarbejderListe, kun i erklæringen og instantieringen af den.

Øvelse 3.6 interface

Lav en klasse Firma, der (bl.a) har en Adresse, ganske som i Medarbejder.

Lav derefter et interface IharAdresse med en property Adresse.

Lad både Medarbejder og Firma inplementere IharAdresse.

Lav om i MedarbejderListe, så den kan indeholde både Medarbejder'e og Firma'er

Test ved også at tilføje Firma'er. Da et firma ikke har et cprnummer eller medarbejdernummer er det nok en god ide at bruge adresse som key

Øvelse 3.7 – Adresse-klasse

Lav en adresse-klasse (med vejnavn, nummer, evt. mere) og tilret adresse-property i Medarbejder, Firma og IharAdresse, så de anvender denne klasse i stedet for en String.

Hvad er fordelen ved dette ved anvendelsen i MedarbejderListe frem for at bruge en String?

Øvelse 3.8 – struct øvelse

Implementer en struct kaldet *Time*, som repræsenterer et tidspunkt 00:00:00-23:59:59. Internt gemmes tidspunktet som et antal sekunder i et privat felt. F.eks. svarer 01:23:20 til 5000 sekunder i det private felt.

Så du kan erklære den sådan : `struct Time {…}`

I de foregående opgaver blev det foreslået at oprette en fil pr. klasse. Her kan I nøjes med at skrive det hele i en cs-fil. Dvs. du kan have din struct defineret i samme namespace som din main metode.

Erklær en int, der indeholder antal sekunder siden midnat i din struct, kald den f.eks. secondsSinceMidnight.

Erklær tre properties, som sætter og returnerer henholdsvis timer, minutter og sekunder (i det følgende antages det, at de hedder Hour, Min og Sek). De skal så også opdatere antal sekunder siden midnat – her et eks. på hvordan Hour kan se ud – prøv at forstå hvad der sker?:

```
public int Hour
            {
                set
                {
                    int hour = value;
                    int temp = _secondsSinceMidnight % 3600;
                    _secondsSinceMidnight = (temp + hour * 3600) % _maxSeconds;

                }

                get { return _secondsSinceMidnight / 3600; }


            }
```

Skriv en constructor, der kan tage tidspunktet som en string på formatet ”hh:mm:ss” (f.eks. ”01:23:20”). Altså altid 3 gange to cifre adskilt af kolon. Du skal så i gang med at bruge substring og parse for at få fat i timer, minutter og sekunder:

Brug SubString() i string klassen til at trække de enkelte værdier ud. Omform til tal med Int32.Parse().

Skriv en anden constructor, der tager 3 int som parametre, nemlig timer, minutter og sekunder og så sætter værdierne

Lav desuden en ToString(), der returnerer tidspunktet på førnævnte format. *Bemærk at signaturen skal indeholde "override", dvs. den skal være*

```
public override string ToString()
```

Test din struct, og prøv f.eks. følgende kode i main:

```
Time t1, t2;
t1=new Time("08:30:00");
t2=t1;
t2.Hour=t2.Hour+2;

Console.Write(t1.ToString());

Console.Write(t2.ToString());
```

Test også fejl i input ved oprettelse og ændring.

Øvelse 3.9 – ekstra frivillig træningsopgave

Byg en Object-orienteret implementation af et poker spil:

- Card deck med 52 standard kort (uden jokers).
- Spillerne – deres navn og hvor mange penge de har tilbage og hvilke kort de har.
- Current Pot - and the amount provided by each player
- Game state

Prøv at implementere de forskellige klasser – du skal overveje hvad de skal have af properties, metoder og constructors.