Objekt Orienteret programmering uge. 1

Dictionaries

Starten på dictionaries var lidt langsom, eftersom vi egentlig bare blev smidt ud i det, men de varede dog ikke længe inden man fandt ud af og bruge *Dictionary*. Sådan som jeg har forstået det, så bruges dictionaries lidt ligesom hvis du ledte i en online ordbog efter et ord, hvor du kunne indsætte en værdi som for eksempel sukker, hvor efter den ville den tage dig til sukkers betydning.

```
public Dictionary<string, int> numbers = new Dictionary<string, int>();
```

Her for oven bliver der lavet en Dictionary, denne dictionary finder en int værdi ud fra en string værdi, det kan du se fordi Dictionarien tager 2 værdier, og den første er din key hvilket i dette eksempel er en string, og valuen er en int. Værdier tilføjes til dictionaries ud fra det navn som du har givet dem, og i dette eksempel er *numbers* navnet på vores dictionary.

```
NRDictionary.numbers["one"] = 1;
```

Tilføjelse af en key og værdi til et dictionary er simpelt nok, du skriver navnet på dit dictionary efterfuldt af brackets, det som skal være inde i de brackets, et det som du vil skrive, for at komme frem til værdien som er 1 i dette tilfælde.

Overloading

Overloading er når u har flere funktioner/mothode eller lignende som hedder det samme, men tager forskellige værdier.

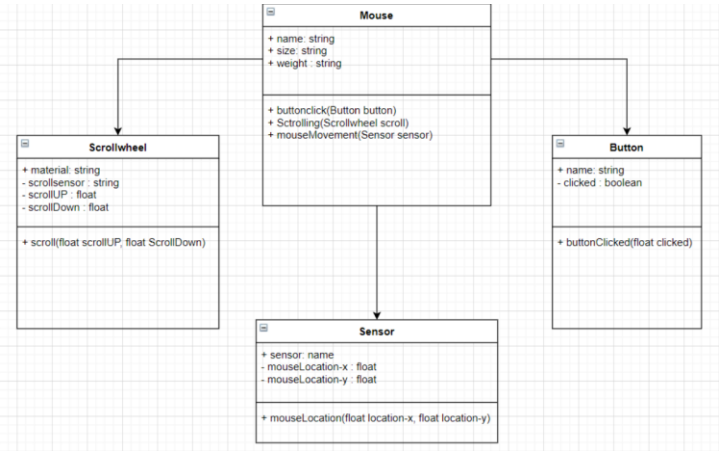
```
public int plus(int a, int b)...

0 references
public float plus(float a, float b)...

0 references
public float plus(string a, string b)...
```

På eksemplet over over, kan du se der er 3 public functions, men da de tager forskellige og eller returner forskellige værdier, så kan de stadig bruges, og det er så det som bliver kaldt *Overloading*. Så i stedet for at lave en ny funktion som hedder

plusInt og en anden der hedder plusFloat, så kalder du bare den begge det samme, men gør så de tager imod forskellige værdier, så når du kalder den, så henter den, den metode/funktion som du kalder, med de værdier som du har givet.

UML-Class Diagrams

Et UML-class diagram giver et overblik over et projekt på forskellige måder, når du har et UML-class diagram over et projekt som lignende.



Det er vigtigt at holde et UML-Class diagram up-to-date, da det er med til at vise de forskellige værdig metoder osv. som du har i dit projekt. Uden et UML-Class diagram så tager en del længere tid at få et overblik over et projekt, især hvis du har mange classes.

Hjemmearbejdsdag d. 06-08-2020

Projektet startede ud fint, startede med at opbygge mit projekt ud fra mit UML-Class diagram som jeg lavede dagen før, fik skrevet det meste ret hurtigt, hvor jeg så finder ud af at der er en del ting som jeg mangler for at skulle kunne have udfyld de forskellige opgaver som vi skulle. Jeg fik så de forskellige tilføjet som manglede, og sørgede for at det virkede som det skulle, og før jeg vidste andet, så var dagen ovre. Følte ikke jeg havde nok tid til projektet, især da jeg gerne ville have haft error handeling med, men selvom jeg ikke havde det med og ikke rigtigt holdte pauser, så sluttede jeg kun lige i tide med at blive færdigt, og også opdatere mit UML-Class diagram så den indeholder de nye ting som jeg havde tilføjet efter jeg begyndte med at skrive koden. Men gennem skrivningen af min kode, fandt jeg et par ting som jeg skal blive bedre til, især hvordan man henter og sætter værdiger af en privat variabel i en klasse.

Delegates

Hvis jeg skal være helt ærlig, så ved jeg ikke rigtigt hvad man bruger delgates til udover at du kan kalde flere metoder/funktioner på en gang og køre dem. Så det virker lidt ligesom et normalt kald på en metode/funktion dog med extra steps.

Lambda Expressions

En anden måde at køre en funktion på, så i stedet for at lave en ny *public void int bla. Bla. bla* så kan du skrive lignende.

```
public delegate int del1(int i1);
del1 handler1 = x => x * x;
Console.WriteLine("Task 1: " + handler1(5) + "\n");
```

Så i stedet for at lave en ny funktion/metode, så skriver jeg hvad jeg vil med det samme, hvilket så er hvad den gør.