Refleksion

Som meget andet er det nok bedst at starte fra starten. Vi startede sidste uge med at lave et personregister som et class library, hensigten med at lave et class library er at gøre koden selvstændig og genanvendelig. Personregisteret som jeg har lavet, beskriver en person med variablerne navn(string), alder(int), arbejde(Job) og hus(Housing). Job og Housing er separate objekter krævet for at oprette en person. Jeg lavede også simple overloads under Job og Housing for hurtigere og eller nemmere oprettelse af flere af objekterne. Generalt var opgaven en rigtig god genopfrisker eftersom jeg ikke har gjort stort brug af objektorienteret programmering siden sidste hovedforløb.

Vi arbejdede også med Dictionaries som er meget brugbare, de er gode for læselighed begrund af deres Key system og er bedre til hastighed end lister eftersom Dictionaries bruger hashkeys til at finde deres data. Et Dictionary er meget som lister i den forstand af at de ikke har given længde og kan teoretisk set være uendelige. For at oprette et Dictionary variabel skal man oprette den med en variabel som bliver dens Key og et andet variabel som bliver hvad Dictionariet indeholder. For at tilføje et entry til et Dictionary kan man enten bruge den indbyggede .Add() funktion eller kalde Dictionariet med [] som indeholder Key valuen og derefter give den variablet. Også vist herunder:

```
Dictionary<float, bool> floatDict = new Dictionary<float, bool>();
floatDict[13] = true;
floatDict.Add(42, false);
```

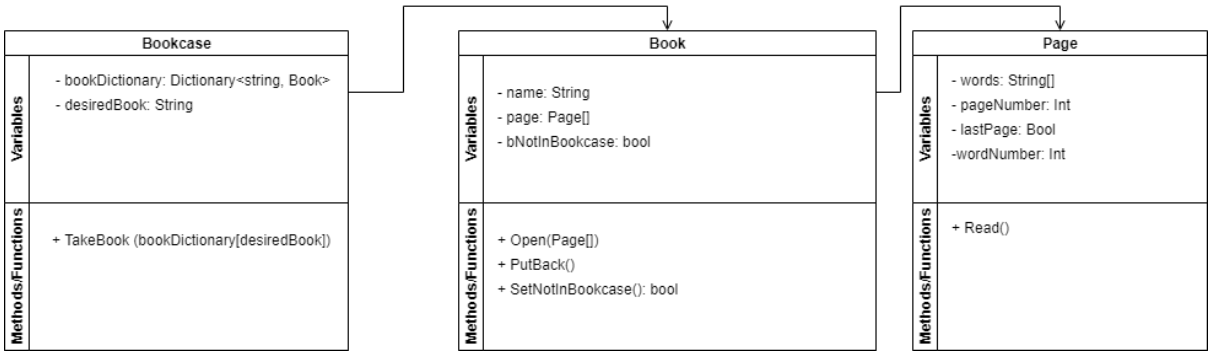
Som en helhed vil jeg sige at Dictionaries er en array type som jeg ser frem til at udnytte mere i fremtiden i mine projekter.

Overloads var en af vores andre emner, overloading er at lave flere funktioner eller metoder med det samme navn som modtager forskellige variabler. Overloading har utrolig mange use cases til fleksibilitet og meget mere. I vores tildelte opgave bliv vi bedt og at lave flere funktioner med det samme navn, hvor nogle af dem modtager floats og andre modtager integers og afhængig af om du caller den med det ene eller andet modtager du forskellige resultater selv hvis at tallene er de samme men gemt som forskellige variabel typer.

```
0 references
public int Plus(int num1, int num2)
{
    return num1 + num2;
}
0 references
public float Plus(float num1, float num2)
{
    return num1 + num2;
}
0 references
public string Plus(string num1, string num2)
{
    int num1S = Int32.Parse(num1);
    int num2S = Int32.Parse(num2);
    string numOutput = $"{num1S + num2S}";
    return numOutput;
}
```

Jeg havde allerede en fin forståelse for overloads og deres use cases men det var stadig rart at få det genopfrisket og at arbejdet lidt mere konkret med dem.

UML-diagrammer gik vi også over, vores opgave var at lave et diagram over et objekt på vores værelse. Jeg valgte at lave et diagram over en bogreol:



den beskriver en bogreol som indeholder bøger der indeholder sider med tekst. UML-diagrammer er et godt redskab til forberedelse af projekter; de hjælper med at visualisere og hjælper også ofte med at identificere problemer før du overhovedet støder på dem.

Torsdag var hjemmestudie dag hvor vi fik tildelt at lave et skolesystem med flere krav f.eks. at kunne få fat i en liste af elever og lærer, se hvilke grupper en bestemt lære underviser og oprette, redigere og fjerne elever, lærer og grupper. Jeg besluttede mig for at give mig selv en ekstra udfordring ved at gøre alle mine variabler private og derved kun direkte anvendelige i deres egen class. Jeg startede som udgangspunkt med at oprette alle mine classes, deres individuelle variabler og derefter metoder som tillod simpel interaktion imellem dem. Efter at have også opsat constructors med nogle simple overloads lavede jeg så en masse funktioner og metoder i min School class:

```
public List<Teachers> GetTeachers()...
1 reference
public List<Students> GetStudents()...
1 reference
public List<Teams> GetTeams()...
0 references
public void AddStudent(string name, Teams team)...
1 reference
public void AddStudent(string name)...
0 references
public void AddTeacher(string name, Teams team)...
0 references
public void AddTeacher(string name)...
1 reference
public void AddTeam()...
```

Navnene giver lidt sig selv men bruges til at hente og lave data i classene under School. For at holde dem så adskilt som muligt bruger mange at funktionerne i School funktioner og metoder i de underliggende classes for at holde så mange variabler private men stadig funktionelle og brugbare. Opgaven blev lidt hård for at veligholde min udfordring om at holde gode code conventioner f.eks. min navngivning til classerne som jeg havde tænkt mig at ændre men blev ved med at udsætte og at beholde alle mine variabler private.

Og sidst men ikke mindst gik vi i gang med Delegates, Delegates bruges til at indeholde metoder og eller funktioner i et variabel. Delegates er meget brugbare til f.eks. køre flere funktioner uden at skulle skrive dem alle sammen ned, man kan også bryde enkapsulering til en hvis grad med public Delegates. Delegates har også endnu en god funktion som hedder Lambda expressions som bruges til at skrive hele funktioner inline:

```
public delegate int DoubleDelegate(int num1);
public delegate float TripleTrouble(float num1, float num2, float num3);
public delegate string HelloDelegate();
0 references
public void TestLambda()
{
    DoubleDelegate doubleThis;
    doubleThis = (int num1) => { return num1 = num1 * 2; };
    doubleThis(2);

    TripleTrouble trouble;
    trouble = (float num1, float num2, float num3) => { return num1 + num2 + num3; };
    trouble(20, 18, 1832);

    HelloDelegate hello;
    hello = () => { return "hello World"; };
    hello();
}
```