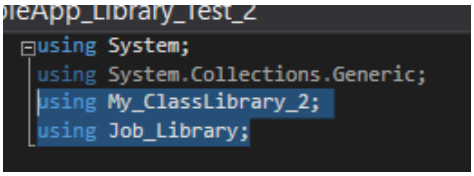
Refleksion h2 ug1 objektorienteret programmering.

Class Library:
Konceptet er efter min mening blot et andet alternative til at arbejde objekt orienteret i c#, hvor man kan referere til de forskellige sider. Der indeholder de classes, du end måtte få brug for, oppe i toppen af siderne, hvor de skal bruges ved ”using” området.



Det gør blandt andet at man kan spare sig en del af de ”class new class” kald, og medfølgende parametre kald, man ellers vil skulle gøre brug af i c# objektorienteret konsol programmering.

```
{
    Player player = new Player();
    TextMethods textMethods = new TextMethods();
    SwitchBoard switchBoard = new SwitchBoard();
    Reward reward = new Reward();
    Obstacles obstacles = new Obstacles();
    BattleEngine battleEngine = new BattleEngine();
    BattleSwitchBoard battleSwitchBoard = new BattleSwitchBoard();
    MonsterObstacle monsterObstacle = new MonsterObstacle();
    EnemyEncounters enemyEncounters = new EnemyEncounters();
    SoundFx soundFx = new SoundFx();
```

```
//TwoPaths Function
switchBoard.TwoPaths(monsterObstacle, switchBoard, obstacles, player,reward, battleSwitchBoard, enemyEncounters, battleEngine, soundFx);
```

Dictionaries:
Kan man bruge i forbindelse med noget list eller array funktion. Til at skaffe mere specifikke data ved brug af nogen Keys, der f.eks. kan være i form af integers eller strings. Der bruges til at relaterer til andet data i f.eks. en liste man har sat op.

```
// Gennemgang af dictionary i klassen anden måde at gøre det på
var ageOfStudents = new Dictionary<string, int>();

ageOfStudents["Tommy"] = 37;
ageOfStudents["Mads"] = 29;
ageOfStudents["Bob"] = 27;
ageOfStudents["Flop"] = 20;
ageOfStudents["Sam"] = 17;
ageOfStudents["Clam Jr"] = 19;
ageOfStudents["Clam"] = 42;
ageOfStudents["Clop"] = 20;
ageOfStudents["Bloat"] = 17;
ageOfStudents["Dex"] = 19;

// Loop over pairs with foreach.
foreach (KeyValuePair<string, int> pair in ageOfStudents)
{
    Console.WriteLine("FOREACH KEYVALUEPAIR: {0}, {1}", pair.Key, pair.Value);
}

Console.WriteLine(ageOfStudents[Console.ReadLine()]);
```

Overloading:
I forbindelse med situationer hvor man kan bruge/genbruge det samme variabel navn optil flere gange i sit program. Ved at tildele den forskellige type værdier såsom string, int og float.

```
2 references
public int Plus(int x, int y)
{
    return x + y;
}

0 references
public float Plus(float x, float y)
{
    return x + y;
}

1 reference
public string Plus(string x, string y)
{
    var result = Int32.Parse(x) + Int32.Parse(y);
    string result_string = result.ToString();
    return result_string;
}
```

UML Diagram:
Et UML-diagram bliver oftest brugt til at planlægge, og få et overblik over et stykke software. Inden selve programmeringsdelen sættes i værk. Det er en del af arbejdes processen, der er dog en del regler og standarder der hører til de forskellige slags diagrammer. Såsom korrekt udseende pile og bokse. Hvilken jeg personligt blot syntes er et irritationsmoment, og går i vejen for et godt flydende workflow. Selve ideen at planlægge visuelt/grafisk går jeg dog ind for, men med knap så meget vigtighed omkring korrekte symboler og pile.

Delegates:
En Delegate er en type, der repræsenterer referencer til nogle metoder med bestemte parametre og returtyper. På nuværende tidspunkt syntes jeg det virker som tilføjet kode for at lave mere kode med relativt samme result. Jeg ved dog godt at der er mere funktion i det end det, men har ikke helt lejet nok med det endnu til helt at kan se det.

```
// Int with parametres
1 reference
public static void MethodWithCallback(int param1, int param2, int param3, Del callback)
{
    callback("The number is: " + (param1 + param2 + param3).ToString());
}
```

```
// Int With parametres console call
MethodWithCallback(1, 2, 3, handler);
```

```
The number is: 6
```