OOP/C# - Opdrachten

Opdrachten bij het vak OOP/C# voor 2e jaars MBO Software Developer Games

Auteur:

René Melenhorst

rgj.melenhorst@noorderpoort.nl

Inhoud

GitHub repository 9

Microsoft Visual Studio 9

1 Opdracht 1 - Hello OOP! 10

1.1 Stap 1 10

1.2 Stap 2 10

1.3 Stap 3 11

2 Opdracht 2 - First steps 12

2.1 Stap 1 12

2.2 Stap 2 12

2.3 Stap 3 13

2.4 Stap 4 14

2.5 Stap 5 14

2.6 Stap 6 14

2.7 Stap 7 15

2.8 Stap 8 15

2.9 Stap 9 16

3 Opdracht 3 - Parents and childs 18

3.1 Stap 1 18

3.2 Stap 2 18

3.3 Stap 3 18

3.4 Stap 4 18

3.5 Stap 5 19

3.6 Stap 6 19
3.7 Stap 7 19
3.8 Stap 8 20
3.9 Stap 9 20
3.10 Stap 10 21
3.11 Stap 11 [Optioneel] 21
4 Opdracht 4 - Only one allowed 23
4.1 Stap 1 23
4.2 Stap 2 24
4.3 Stap 3 24
4.4 Stap 4 24
4.5 Stap 5 25
4.6 Stap 6 26
4.7 Stap 7 26
4.8 Stap 8 26
4.9 Stap 9 27
4.10 Stap 10 [Optioneel] 27
5 Opdracht 5 - Props 28
5.1 Stap 1 28
5.2 Stap 2 29
5.3 Stap 3 29
5.4 Stap 4 31
5.5 Stap 5 32

5.6 Stap 6 33
5.7 Stap 7 34
5.8 Stap 8 [Optioneel] 35
6 Opdracht 6 - Relations… (need I say more?) 36
6.1 Stap 1 37
6.2 Stap 2 37
6.3 Stap 3 [Optioneel] 39
7 Opdracht 7 - Yet another relation 40
7.1 Stap 1 40
7.2 Stap 2 42
7.3 Stap 3 42
7.4 Stap 4 43
7.5 Stap 5 43
7.6 Stap 6 43
7.7 Stap 7 [Optioneel] 44
7.8 Stap 8 [Optioneel] 44
8 Opdracht 8a - The real deal (database interaction) – Part 1 Setup 46
8.1 Stap 1 46
8.2 Stap 2 47
8.3 Stap 3 47
8.4 Stap 4 48
8.5 Stap 5 49
8.6 Stap 6 49

8.7 Stap 7 50
8.8 Stap 8 50
8.9 Stap 9 51
Opdracht 8b - The real deal – Part 2 Implementation 52
8.10 Stap 1 52
8.11 Stap 2 52
8.12 Stap 3 53
8.13 Stap 4 - CRUD 54
8.14 Stap 5 - Create 54
8.15 Stap 6 - Read 55
8.16 Stap 7 - Update 56
8.17 Stap 8 - Delete 57
9 Opdracht 9 - Zie Scherp Scherper H9 58
GitHub repo en MS Visual Studio 58
9.1 Stap 1 - OOP Intro 59
9.2 Stap 2 - Klassen en objecten in C# 60
9.3 Stap 3 - Methoden en access modifiers 61
9.4 Stap 4 - Properties 63
9.5 Stap 5 - DateTime: leren werken met objecten 64
9.6 Laatste stap - Oefeningen 65
10 Opdracht 10 - Zie Scherp Scherper H10 75
GitHub repo en MS Visual Studio 75
10.1 Stap 1 - Stack en Heap 76

10.2 Stap 2 - Objecten en methoden 77

10.3 Stap 3 - Null en NullReferenceException 78

10.4 Stap 4 - Namespaces en using 79

10.5 Stap 5 - Exception handling 80

10.6 Laatste stap - Oefeningen 82

11 Opdracht 11 - Zie Scherp Scherper H11 87

GitHub repo en MS Visual Studio 87

11.1 Stap 1 - Constructors en de default Constructors 87

11.2 Stap 2 - Overloaded Constructors 88

11.3 Stap 3 - Object Initializer Syntax 90

11.4 Stap 4 - Static 91

11.5 Laatste stap - Oefeningen 93

12 Opdracht 12 - Zie Scherp Scherper H12 98

GitHub repo en MS Visual Studio 98

12.1 Stap 1 - Arrays van objecten 98

12.2 Stap 2 - List 99

12.3 Stap 3 - Foreach en var 101

12.4 Stap 3 - Nuttige collection klassen 102

12.5 Laatste stap - Oefeningen 103

13 Opdracht 13 - Zie Scherp Scherper H13 114

GitHub repo en MS Visual Studio 114

13.1 Stap 1 - Overerving intro 114

13.2 Stap 2 - Constructors bij overerving 115

13.3 Stap 3 - Virtual en override 117

13.4 Stap 4 - Base keyword 118

13.5 Laatste stap - Oefeningen 118

14 Opdracht 14 - Zie Scherp Scherper H14 122

GitHub repo en MS Visual Studio 122

14.1 Stap 1 - System.Object 122

14.2 Stap 2 - Abstract 123

14.3 Stap 3 - Eigen exceptions maken 124

14.4 Laatste stap - Oefeningen 125

15 Opdracht 15 - Zie Scherp Scherper H15 133

GitHub repo en MS Visual Studio 133

15.1 Stap 1 - Compositie 133

15.2 Stap 2 - this keyword 135

15.3 Laatste stap - Oefeningen 135

16 Opdracht 16 - Zie Scherp Scherper H16 141

GitHub repo en MS Visual Studio 141

16.1 Stap 1 - Polymorfisme 141

16.2 Stap 2 - Polymorfisme in de praktijk 142

16.3 Stap 3 - 'Is' en 'As' keywords 144

16.4 Stap 4 - Objecten vergelijken 145

16.5 Laatste stap - Oefeningen 146

17 Opdracht 17 - Zie Scherp Scherper H17 149

GitHub repo en MS Visual Studio 149

17.1 Stap 1 - Interface intro 149

17.2 Stap 2 - Interfaces voorbeeld 150

17.3 Stap 3 - Interfaces in de praktijk 152

17.4 Stap 4 - Interfaces en polymorfisme 153

17.5 Laatste stap - Oefeningen 155

18 Opdracht 18 - Zie Scherp Scherper Appendix [Optioneel] 160

GitHub repo en MS Visual Studio 160

18.1 Stap 1 - Out en Ref parameters 160

18.2 Stap 2 - TryParse 161

18.3 Stap 3 - Operator overloading (TBD) 162

Bijlage A - Een project reference toevoegen 163

GitHub repository

Alle opdrachten voor dit blok maak je in één repository. Dus je gaat straks een lokale repo maken en koppelen aan de repo behorende bij de volgende GitHub Classroom assignment:

https://classroom.github.com/a/caTj2Dr9

Accepteer bovenstaande assignment.

Navigeer in GitHub naar je nieuwe repo en controleer of de 'Quick setup' pagina actief is. Indien het geval volg dan de stappen die beschreven staan in het document **OOP-C#-jaar2-blok5-GitHub-repo.pdf**, vanaf het kopje 'GitHub desktop'.

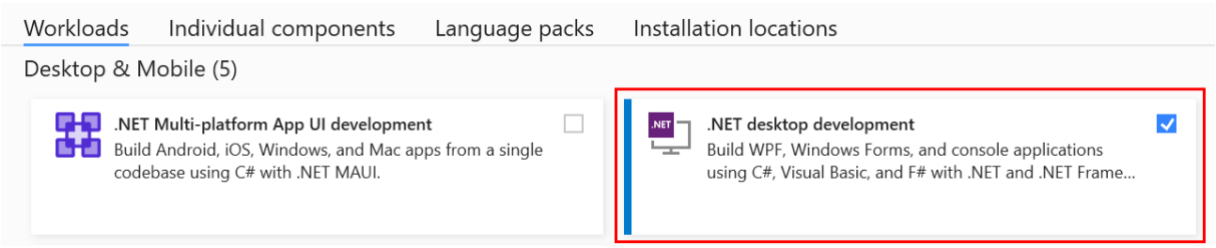
Microsoft Visual Studio

Controleer of op je laptop een versie van *Microsoft .NET* is geïnstalleerd, welke versie is niet zo van belang voor deze tutorial. Indien .NET niet is geïnstalleerd, installeer dan de laatste versie:
https://learn.microsoft.com/en-us/dotnet/core/install/how-to-detect-installed-versions?pivots=os-windows

https://learn.microsoft.com/en-us/dotnet/core/install

Installeer daarna de laatste versie van *Microsoft Visual Studio* ***Community Edition***:
https://visualstudio.microsoft.com/vs/community

Zorg ervoor dat je tijdens de installatie het component *.NET desktop development* selecteert:

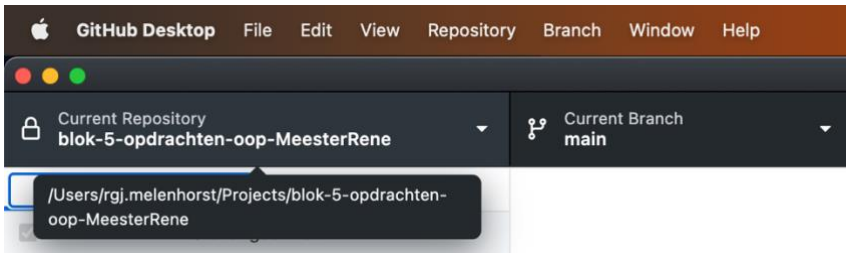


1 Opdracht 1 - Hello OOP!

In deze opdracht maken we in Visual Studio de eerste versie van het OOP project dat we in de volgende opdrachten steeds verder gaan uitbreiden.

1.1 Stap 1

Maak in Visual Studio een nieuw C# project met de naam **OOP** gebaseerd op de template 'Console Application C#'. Kies als locatie voor het project dezelfde folder als de plek waar ook je **lokale GitHub repo** voor deze les staat. Zie ook *GitHub Desktop* als je vergeten bent waar die staat (hover met je muis over de repo naam):



1.2 Stap 2

Nadat het nieuwe project is aangemaakt zie je in de solution explorer de OOP solution directory staan met daarin de project directory (met dezelfde naam), en daarin de file *Program.cs*.

Afhankelijk van de .NET versie die je gebruikt bestaat de inhoud van deze C# file uit:

1. òf een *namespace* met daarin een *class*, met daarin een *Main() method* met een 'Hello World' print statement (*Console.WriteLine*).
2. òf alleen het bovengenoemde print statement.

Dit verschil wordt uitgelegd in de onderstaande link:

https://aka.ms/new-console-template

Voor ons OOP project willen wij **de eerste optie** als basis gebruiken, dus als in jouw geval de tweede optie geldt, lees dan de uitleg in de link hierboven even door.

Update daarna, aan de hand van het beschreven voorbeeld in de link, de code van *Program.cs* zodat die begint met een *namespace* genaamd **OOP**, met daarin een class genaamd **Program**, met daarin een *public static void Main() method* met het reeds bestaande print statement.

Voor de MS Windows gebruikers (voor Mac overbodig): Voeg onder het 'Hello World' print statement een *Console.ReadKey()* statement toe zodat het *Output Window* blijft staan (totdat je op een key drukt) en je het print statement, en in de toekomst andere debug statements, kunt controleren voordat het *Output Window* weer automatisch verdwijnt.

1.3 Stap 3

Test je nieuwe project door het te bouwen en te runnen. Als alles goed is gegaan wordt het 'Hello World' print statement naar het Terminal Window in Visual Studio geschreven. Is dit inderdaad het geval dan wordt het tijd om voor de eerste keer je code te *committen* en naar GitHub te *pushen* met behulp van GitHub Desktop.

Gebruik als commit message **Opdracht 1** en controleer na het laatste commando, de *Git push*, of je project aan je remote GitHub repository is toegevoegd.

2 Opdracht 2 - First steps

Bestudeer voor deze opdracht de Reader OOP, hoofdstuk 1 'Object - een nieuw type variabele'.

In deze opdracht maken we onze eerste klasse met methods en properties. We gaan een applicatie maken waarmee we game gerelateerde gegevens uit een database kunnen lezen. Maar voordat we gaan communiceren met een echte database, gaan we in de volgende opdrachten gebruik maken van *Dictionaries* en *KeyValuePairs* om de gamegegevens in op te slaan.

2.1 Stap 1

Maak in de project directory een nieuwe folder genaamd **classes** met daarin de class **Game**.

Maak in de Game class een *private* property met de naam **connection** van het type **SqlConnection**. Deze .NET class is onderdeel van de *System.Data.SqlClient* namespace dus voeg een *using statement* toe en zeer waarschijnlijk moet je ook de bijbehorende package m.b.v. de *NuGet package manager* installeren, zie ook:

https://www.mytecbits.com/microsoft/dot-net/error-sqlconnection-could-not-be-found

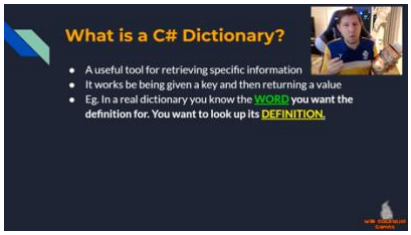
Vooralsnog gaan we niets met deze connectie doen, dus meer uitleg volgt later, maar zorg in ieder geval dat de IDE het type herkent in de code, en dat de Game class goed bouwt.

2.2 Stap 2

We gaan een property definiëren die gebruikt gaat worden om de database-records van de tabel games op te slaan, en wel in de vorm van een *associatieve array*. In C# staat de associatieve array ook wel bekend als een *dictionary*.

Lees, en eventueel kijk (over dictionaries in Unity!) daarom eerst de onderstaande uitleg over de dictionary. Kijk goed hoe je een dictionary definieert, vult met waardes en ook hoe je de data in een dictionary weer uitleest.

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h12-arrays-en-klassen/dict#dictionary-less-than-greater-than-collectie



Klik op het plaatje om de video tutorial te bekijken. Of als je dit document in een browser bekijkt gebruik dan deze link: https://www.youtube.com/watch?v=Cyj60xwhvb8

In het kort: in een dictionary wordt ieder element voorgesteld door een sleutel (*key*) en de waarde (*value*) van het element. De sleutel moet een unieke waarde zijn zodat het element kan opgevraagd worden uit de dictionary aan de hand van deze sleutel zonder dat er duplicaten zijn. Bij de declaratie van de dictionary dien je op te geven wat het datatype van de key zal zijn, alsook het type van de value.

Maak in de Game class een *private* property **table_data** van het type Dictionary waarbij de *key* van het type **int** is en de *value* van het type **string**.

2.3 Stap 3

In de volgende stappen gaan we in de Game class een method genaamd *find()* maken die we gaan gebruiken om een enkel item uit een dictionary te halen (lees: een record uit een tabel). Voor dit item gaan we het type *KeyValuePair* gebruiken, aangezien de elementen in een dictionary worden opgeslagen als *KeyValuePair* objecten.

Lees de onderstaande uitleg over dit type:

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.keyvaluepair

Het return type van de *find()* method wordt zodoende een **KeyValuePair<int, string>**. Zoals je ziet zijn de types voor de *key* (int) en de *value* (string) dezelfde als die van de dictionary.

2.4 Stap 4

Voordat je met de *find()* method aan de slag gaat, maak je in de Game class een *private* property **record** van het type **KeyValuePair** waarbij de *key* van het type **int** is en de *value* van het type **string**.

Deze property gaan we gebruiken om een enkel record uit de *table_data* Dictionary op te slaan, nadat we het record hebben opgezocht in de tabel.

2.5 Stap 5

In de find() method komt (in de toekomst) de code die je nodig hebt voor het lezen van één databaserecord uit de tabel *games*. Op dit moment gaan we de dictionary gebruiken om het lezen uit een databasetabel te simuleren.

Hiervoor treffen we de volgende voorbereidende stappen in de **constructor**:

1. Vul de **table_data** property (van het type dictionary) met een key/value waarde waarbij de key 1 is en de value 'Battlefield 2042'.
2. Vul de dictionary aan met enkele (minstens twee) andere zelfgekozen games, met opeenvolgende waardes voor de key.

2.6 Stap 6

Vervolgens implementeren we de find() method:

1. Maak in de Game class de method **find()**, visibility *public*, met één parameter: **id** van het type int. Het return type van de method wordt **KeyValuePair<int, string>**.
2. Maak in deze method een lokale variabele **record** en stop hierin de key/value combinatie uit de dictionary waarvan de key overeenkomt de input parameter *id* (nadere uitleg onder).
3. Sla de waarde van deze variabele op in de gelijknamige class property (this.record).
4. Laat de method de waarde van de lokale variabele retourneren.

Betreffende stap 2: Het type voor de lokale variabele **record** is dus dezelfde als van het *return type* van de method, d.w.z. de KeyValuePair. Je maakt van dit data type een variabele

aan (keyword *new*) en maakt de waarde van de *key* gelijk aan de input parameter *id*, en de waarde van *value* gelijk aan de bijbehorende waarde van het dictionary element.

Vind je deze stap lastig maak dan, voordat je de record variabele vult, eerst een hulp variabele, b.v. genaamd **key**, die je gelijk stelt aan de input parameter *id*. Daarna maak je een tweede hulp variabele, b.v. genaamd **value**, die je vult met de bijbehorende waarde van het dictionary element. Nu je zowel de *key* als de *value* waarde hebt, kun je deze gebruiken om de record van het type *KeyValuePair* te vullen.

Controleer of je applicatie na deze aanpassingen nog steeds goed bouwt. Zo ja, commit en push je de code naar je GitHub repo met als message: **Opdracht 2a**.

2.7 Stap 7

Maak een public getter method met de naam **getTableData**. Deze getter moet de waarde van de property **this.table_data** retourneren. Denk erom de goede *return type* voor de method te definiëren, die is dezelfde als het type van de dictionary.

Maak een public getter method met de naam **getRecord**. Deze getter moet de waarde van de property **this.record** retourneren. Denk erom de goede *return type* voor de method te definiëren, die is dezelfde als het type van de KeyValuePair.

2.8 Stap 8

Maak een public setter method met de naam **setTableData**. Deze setter moet de waarde van de property **this.table_data** toekennen. Denk erom de goede *input parameter type* voor de method te definiëren, die is dezelfde als het type van de dictionary.

Maak een public setter method met de naam **setRecord**. Deze setter moet de waarde van de property **this.record** toekennen. Denk erom de goede *input parameter type* voor de method te definiëren, die is dezelfde als het type van de KeyValuePair.

Nu je een setter hebt voor het opslaan van het record property, kun je de bestaande schrijfactie in de *find()* method ook vervangen door de aanroep van de setter. Doe dat.

2.9 Stap 9

Het wordt nu tijd om je code te testen. Het eenvoudigst is dat om dat in de Main() method van het hoofdprogramma te doen, oftewel in de Program class.

Voor elke nieuwe opdracht maken we een nieuwe *private static void* method, in dit geval genaamd **Opdracht2()**, die je vervolgens in de Main() method aanroept. Zo blijft je test code, ook na het maken van de volgende opdrachten, per opdracht overzichtelijk bij elkaar staan.

Maak de method Opdracht2() in de Program class en voeg code toe die:

- Een nieuw object maakt van de class Game, en noem dat object **game**.
- De find() method aanroept op het game object, met als parameter een geldige id (zie daarvoor de inhoud van de dictionary). Het resultaat stop je in een variabele, waarbij **record** een goede keuze zou zijn omdat je 1 game record ophaalt.
- De inhoud van de variabele toont d.m.v. *console writeline*. Zowel de key als de value moeten zichtbaar worden d.m.v. 1 statement. Zorg dat je de Key en Value waardes apart ophaalt uit het *KeyValuePair* game record.
- De getTableData() method aanroept op het game object en het resultaat in een variabele stopt. Omdat je meerdere games records ophaalt uit de dictionary, is **games** een goede keuze.
- De inhoud van de *games* variabele toont d.m.v. een foreach-loop en een *console writeline*. Voor elke *game* moeten zowel de key als de value zichtbaar worden d.m.v. 1 statement. Ook hier zorg dat je de Key en Value waardes apart ophaalt uit elke *KeyValuePair* game record.

[Tip] Gebruik *code snippet* 'cw' als je een console writeline nodig hebt (2x tab tijdens of na het typen van het woord). Zie deze link voor een volledige lijst:
https://learn.microsoft.com/en-us/visualstudio/ide/visual-csharp-code-snippets

Test je nieuwe project door het te bouwen en te runnen. Controleer of de print statements de verwachte output genereren, dus zoiets als:

```
> Record found:
[1] Battlefield 2042

> Games table:
[1] Battlefield 2042
[2] FIFA 22
[3] Dirt Rally 2.0
```

Test je code ook door er met de debugger stap voor stap doorheen te lopen, b.v. door gebruik te maken van *breakpoints*, en naar de waardes van de gebruikte lokale variabelen te kijken. Zoals in onderstaand voorbeeld met een breakpoint in de *find()* method, vlak voor het verlaten van de method.

Locals | Search Results

Name	Value	Type
this	{OOP.Game}	OOP.Game
connection	{System.Data.SqlClient.SqlConnection}	System.Data.SqlClient.SqlConnection
@record	{System.Collections.Generic.KeyValuePair<int, string>}	System.Collections.Generic.KeyValuePair<int,string>
Key	1	int
Value	"Battlefield 2042"	string
table_data	Count = 3	System.Collections.Generic.Dictionary<int,string>
[0]	{System.Collections.Generic.KeyValuePair<int, string>}	System.Collections.Generic.KeyValuePair<int,string>
Key	1	int
Value	"Battlefield 2042"	string
[1]	{System.Collections.Generic.KeyValuePair<int, string>}	System.Collections.Generic.KeyValuePair<int,string>
[2]	{System.Collections.Generic.KeyValuePair<int, string>}	System.Collections.Generic.KeyValuePair<int,string>
Raw View		
id	1	int
record	{System.Collections.Generic.KeyValuePair<int, string>}	System.Collections.Generic.KeyValuePair<int,string>?

Test de *find()* method ook met een **ongeldige id**, en probeer het probleem wat ontstaat op te lossen. Geef feedback als het record niet bestaat, dus zoiets als 'Record does not exist'.

Indien je test succesvol is verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 2b**.

3 Opdracht 3 - Parents and childs

Bestudeer voor deze opdracht Reader OOP, hoofdstuk 2 'Extending classes'.

In deze opdracht maak je een tweede klasse die bij een databasetabel hoort, en je maakt een *abstract parent class* voor beide classes, waarin je gemeenschappelijke code plaatst.

3.1 Stap 1

Maak een nieuwe class **Publisher** in de directory **classes**, en hanteer dezelfde *naming convention* voor de filenaam als je bij de class Game hebt gedaan. Je hoeft er nog geen nieuwe code in te zetten.

3.2 Stap 2

Maak in dezelfde directory een nieuwe **abstract** class **Model**. Zorg ervoor dat zowel de classes Game als Publisher **child classes** worden van de class Model.

3.3 Stap 3

Bedenk dat je nieuwe class Publisher net als de class Game (in de toekomst) een database connectie nodig heeft. Plaats hiervoor echter GEEN nieuwe de code in de class Publisher, maar **verplaats** de property **connection** van de inmiddels child class Game naar de parent class Model.

Zorg ervoor dat de property *connection* beschikbaar is in de child classes, maar NIET als *public* property! (zie de reader)

Je test code uit de vorige opdracht zou nog steeds goed moeten werken. Check dat!

3.4 Stap 4

Voer nu dezelfde acties uit met de properties **table_data** en **record** en de bijbehorende **getters** en **setters**. Ook hier geldt: properties *table_data* en *record* moet beschikbaar zijn in de child classes, maar niet public!

Ook nu weer: check of je test code nog steeds goed werkt.

3.5 Stap 5

Voeg aan de constructor van het Publisher class code toe om, op dezelfde manier als in de Game class, de *table_data* property met een key/value waarde te vullen waarbij de key 1 is en de value 'Electronic Arts'.

Vul de dictionary aan met enkele (minstens twee) andere zelfgekozen publishers, met opeenvolgende waardes voor de key.

3.6 Stap 6

Je hebt nu in de Game class, behalve de constructor, alleen nog maar de method *find()* over. Als je die bekijkt en bedenkt hoe de vergelijkbare method er in de class Publisher uit zou moeten zien, dan is er maar één verschil: de inhoud van de databasetabel, oftewel de *table_data* dictionary. In het ene geval: *games* gerelateerde tabeldata, en in het andere geval *publisher* gerelateerd.

Echter omdat we op dit moment nog geen data uit een 'echte' database halen hoeven we nu ook nog geen onderscheid te maken tussen tabelnamen. Dus dat nodigt uit om de *find()* method generiek te maken en óók te verplaatsen naar de parent class, en de naam van *table_data* ongewijzigd te laten.

Doe dat en check of je test code daarna nog steeds goed werkt.

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 3a**.

3.7 Stap 7

We gaan een volgende voorbereidende stap nemen in ons doel uiteindelijk met een 'echte' database te communiceren. In dat geval moeten we in de Model class wél onderscheid kunnen maken uit welke tabel we de data willen lezen, namelijk de tabel *games* of *publishers*.

Voeg daarom de protected property **tablename** van het type **string** toe aan de Model class.

3.8 Stap 8

Je hebt nu twee min of meer lege child classes (Game, Publisher) en een property *tablename* in de parent (Model) van die classes zonder waarde. Dat wordt een taak voor de child classes. We gaan de constructor van de child classes aanpassen: die moeten nu de constructor in de parent **overriden**! (zie reader).

De nieuwe child class constructors krijgt twee taken:

1. De parent constructor van de Model class aanroepen.
2. Aan de parent constructor de waarde van de desbetreffende tablename doorgeven d.m.v. van een parameter.

In de Model class zelf komt via deze parameter de waarde van de tablename binnen, en daar moet je dus de juiste waarde aan de property *tablename* toekennen (this.tablename = …).

Pas na de bovenstaande aanpassingen je test code ook aan, want de Game constructor verwacht nu een parameter. Verzin een logische naam voor de tablename die je doorgeeft.

Als je het goed hebt gedaan, moet de tests code nog steeds werken.

3.9 Stap 9

Voeg code toe aan je test om ook de gegevens van een publisher te lezen en te tonen. Hiervoor maak je in de *Program* class een nieuwe test method, in dit geval genaamd **Opdracht3()**, die je vervolgens in de Main() method aanroept.

Deze stap begint dus met het maken van een object van de class Publisher. Zie het test voorbeeld van de Game class voor de toe te passen *naming conventions* en de vervolgstappen.

Daarna maak je gebruikt van de *find()* method om een specifiek record te zoeken, en van de *getTableData()* method om alle records op te halen. Toon in beide gevallen de resultaten aan de gebruiker.

Controleer of de print statements de verwachte output genereren, dus zoiets als:

```
> Record found:
[1] Electronic Arts

> Publishers table:
[1] Electronic Arts
[2] Codemasters
[3] Nintendo
```

Stap als onderdeel van de test ook met de debugger door de code vanaf de instantiatie van de class in de test method, via de child class naar de parent class en weer terug, en bekijk waarde van de local variables in de tussentijd.

Test de *find()* method ook met een ongeldige id, en geef net als bij de test van de Game class feedback als het record niet bestaat.

Indien je test succesvol is verlopen, commit en push je de code naar je repo met als message: **Opdracht 3b**.

3.10 Stap 10

Voeg code toe aan je test om een object te maken van de class Model. Als je het goed hebt gedaan, krijg je een foutmelding:

***Error**: Cannot create an instance of the abstract type or interface 'Model'*

Het is niet nodig deze test code te committen, maar je zou inmiddels goed moeten begrijpen waarom dit niet kan!

3.11 Stap 11 [Optioneel]

Het *id* van het record dat je in je test code opzoekt met de *find()* method geef je nu hard-coded (oftewel: *compile-time*) aan. Wil je een ander record opzoeken, incl. een niet bestaande, dan moet je telkens de code aanpassen en opnieuw bouwen voordat je de nieuwe test kunt runnen. Dat is natuurlijk niet erg flexibel.

Pas je test code aan zodat je *run-time* het id van het op te zoeken record bepaald, b.v. op één van onderstaande manieren:

- Voor de aanroep van *find()* method declareer je een nieuwe variabele die je een **random integer** waarde geeft. Deze variabele geef je vervolgens mee als parameter aan de *find()* method.
  Zorg voor een random range die buiten de key range van je dictionary valt, zodat je automatisch ook het opzoeken van niet bestaande records test.
- Vraag d.m.v. **user input** de waarde van het te zoeken record op. Check voordat je de *find()* method aanroept of de gebruiker een integer waarde heeft ingetypt.
  Zorg bij het runnen van de test dat je ook waardes opgeeft die buiten de key range van je dictionary vallen, zodat je ook het opzoeken van niet bestaande records test.
  En test met niet-integers, oftewel een char of string.

Na de aanpassing run je test code een aantal malen, waarbij er dus steeds een ander record wordt opgezocht zonder dat je de code tussendoor aanpast.

Indien je test succesvol is verlopen, commit en push je de code naar je repo met als message: **Opdracht 3c**.

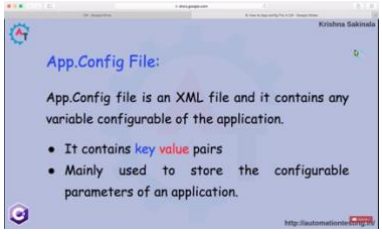
4 Opdracht 4 - Only one allowed

Bestudeer voor deze opdracht Reader OOP, hoofdstuk 3 'Static en singleton'.

In deze opdracht maak je twee *singleton classes*, één voor het lezen en beschikbaar maken van configuratiegegevens, en één voor het beschikbaar maken van de connectie met de database.

De configuratiegegevens die we gaan opslaan in de singleton, gaan we inlezen vanuit een configuratie file, om precies te zijn een XML bestand. Lees, en eventueel kijk, daarom eerst de onderstaande uitleg over hoe je een XML bestand uitleest:

https://docs.microsoft.com/en-us/troubleshoot/developer/visualstudio/csharp/language-compilers/store-custom-information-config-file



Klik op het plaatje om de video tutorial te bekijken. Of als je dit document in een browser bekijkt gebruik dan deze link: https://www.youtube.com/watch?v=IFIiKct4OoE

Je hoeft de genoemde stappen niet direct uit te voeren, dat gaan we hieronder stap voor stap doen, maar lees ze wel goed door zodat je bij elke onderstaande stap weet waar het gevraagde staat beschreven.

4.1 Stap 1

Maak een XML configuratie file **App.config** (file type 'Empty XML file') in de project root (daar waar het hoofdprogramma, oftewel de Program class, ook staat). De inhoud van de file bestaat uit databasegegevens en moet er als volgt uitzien, in XML formaat:

```xml
<?xml version="1.0" encoding="UTF-8" ?>

<configuration>
    <appSettings>
        <add key="driver" value="mysql" />
        <add key="host" value="localhost" />
        <add key="port" value="3306" />
        <add key="database" value="project_games" />
        <add key="username" value="noorderpoort" />
        <add key="password" value="1234" />
    </appSettings>
</configuration>
```

Voeg dit bestand toe aan je .gitignore.

Zoals je ziet worden de configuratie items in een *key/value* formaat opgeslagen, net als in een dictionary, het data type waarmee je in het vorige hoofdstuk kennis mee hebt gemaakt.

4.2 Stap 2

Maak in de classes directory een nieuwe class met de naam **Config** en neem daarin de singleton standaardstructuur uit de reader over.

4.3 Stap 3

Voeg een private property **contents** toe, van het type Dictionary waarbij de *key* en de *value* van het type **string** zijn. Niet static!

4.4 Stap 4

Voeg aan de constructor de code toe die de property contents vult met de waardes uit de configuratiefile. In pseudo code wordt dat dus zoiets als hieronder voor 1 configuratie item:

```
string key = "driver"
string value = <haal de driver value op m.b.v. de ConfigurationManager>
<voeg de key/value pair toe aan de contents property>
```

Wat je nodig hebt om items uit de file te lezen heb je in het begin van dit hoofdstuk gezien. Lees daarom deze stappen nog even terug op de internet pagina, en/of kijk de tutorial video nogmaals.

In het kort: voeg de benodigde using statements toe aan de Config class file, voeg de *System.Configuration.ConfigurationManager* package toe als een project reference, en zorg dat je de contents dictionary correct gevuld krijgt.

Test het correct uitlezen van de configuratie file met (tijdelijke) print statements in de constructor die de inhoud van de dictionary in de console laat zien. De inhoud van een dictionary printen doe je al in de Program class, dus deze code kun je als voorbeeld gebruiken!

Of nog beter: gebruik de debugger, zet een breakpoint in de constructor, en bekijk de waardes die je uitleest en in de dictionary zet.

Hiervoor maak je in de *Program* class een nieuwe test method, in dit geval genaamd **Opdracht4()**, die je vervolgens in de Main() method aanroept. Maak een instantie van de Config class, en doe dit op de *singleton manier*! (zie reader)

Indien je test succesvol is verlopen, commit en push je de code naar je GitHub repo met als message: **Opdracht 4a**.

4.5 Stap 5

De class moet op twee manieren gebruikt kunnen worden voor het opvragen van configuratiegegevens:

```
var section = Config.getInstance().section();
var pw = Config.getInstance().item("password");
```

De eerste method *section()* retourneert een associatieve array, de dictionary dus, en wel de gehele dictionary. De tweede method *item()* retourneert een specifiek item uit de dictionary.

Het *return type* van de eerste method wordt een dictionary van hetzelfde type als de contents property, en het return type van tweede method wordt een **KeyValuePair**. Zie ook het vorige hoofdstuk waarin je dit data type ook hebt geretourneerd, namelijk in de *find()* method.

Implementeer deze twee methods in de Config class (denk erom: niet static!).

4.6 Stap 6

Test je nieuwe methods uit de configuratie class in je hoofdprogramma (!) m.b.v. het uitvoeren van de twee statements uit de vorige stap en print het resultaat naar de console, dus zoiets als:

```
> Config section:
driver - mysql
host - localhost
port - 3306
database - project_games
username - noorderpoort
password - 1234
> Config item:
[password] 1234
```

Test de aanroep van de *item()* method ook met een ongeldige key, en probeer het probleem wat ontstaat op te lossen. Dit heb je als het goed is al eerder gedaan voor de *find()* method, die hetzelfde *return type* heeft, dus je kunt hier dezelfde oplossing implementeren.

Als je met behulp van de *section()* method alle configuratie items succesvol op je scherm kunt tonen, kun je het print statement in de constructor in commentaar plaatsen (of weghalen), dit doe je namelijk nu dubbel.

Indien je test succesvol is verlopen, *commit* en *push* je de code naar je repo met als message: **Opdracht 4b**.

4.7 Stap 7

Maak een nieuwe singleton class **Database**. Neem weer de **singleton** standaardstructuur over.

4.8 Stap 8

Verplaats de *protected* (want in een *parent class*) property **connection** van het class type **SqlConnection** van de Model class naar de Database class, en verander de visibility in *private* (want in een *singleton class*). Verplaats ook de *using statement* mee, al gebeurt dit waarschijnlijk al automatisch.

4.9 Stap 9

In de constructor komt (later) de code die je nodig hebt om een 'echte' databaseconnectie te maken. Op dit moment treffen we slechts een voorbereidende stap:

Haal in de Database constructor met behulp van de Config class de database configuratie gegevens op en print (als test) de items in een foreach loop. Hiervoor moet je natuurlijk ook , net als in je test method, zorgen dat de *getInstance()* method wordt aangeroepen.

Maak in je test method in de *Program* class een instantie van de Database class, en doe dit net als voor de Config class op de *singleton manier*! Als je vervolgens in de Database constructor alle configuratie items succesvol op je scherm kunt tonen kun je de foreach loop in commentaar plaatsen (of weghalen), deze waardes printen doe je namelijk ook al in de test code van de *Config* class.

Indien je test succesvol is verlopen, *commit* en *push* je de code met als message: **Opdracht 4c**.

4.10 Stap 10 [Optioneel]

Nu we een database klasse hebben, is het netjes om de *table_data* property hier ook naartoe te verplaatsen. Eigenlijk hoort die property (met de data van alle games of publishers), in tegenstelling tot de *record* property (met de data van 1 game of publisher), natuurlijk ook niet thuis in de Model class (die 1 game of publisher *modelleert*).

Doe dat en zorg dat je in de Model parent class en de child classes nog steeds toegang hebt tot de table data. Bedenk zelf wat hiervoor nodig is, gebruik makende van wat je in de vorige stappen hebt geleerd. Maak overal waar je (alle) table data nodig hebt gebruik van deze nieuwe property.

Verplaats nu ook de *tablename* property van de Model class naar de Database class, want deze hoort logischerwijs bij de table_data property. Maak overal waar je de tablename nodig hebt gebruik van deze nieuwe property.

Test je code en indien succesvol verlopen *commit* en *push* je de code met als message: **Opdracht 4d**.

5 Opdracht 5 - Props

Je gaat in deze opdracht *full properties* en *auto properties* inbouwen en een deze in het hoofdprogramma gebruiken. Bestudeer voor deze opdracht hoofdstuk 4 'Full- en auto properties' van de reader.

Met onze huidige type definitie van de dictionary *table_data*, waarbij de *value* bestaat uit de *scalar type* **string**, kunnen we slecht 1 tabelveld opslaan, in ons geval de naam van de game of publisher. Bij een database-tabel is er echter in de meeste gevallen sprake van meer dan 1 veld, zo ook in ons geval.

We zullen om deze data ook in een dictionary op te kunnen slaan dus gebruik moet maken van een ander value type, en wel het *compound type* **object**. Daarom gaan we de type definitie van de dictionary aanpassen.

5.1 Stap 1

Maak in de parent class file Model.cs, vlak boven de Model class definition zelf, maar wel in dezelfde namespace, een nieuwe *public abstract class* genaamd **ModelRecord**. Deze klasse gaan we gebruiken om de record informatie op te slaan die in beide child classes voorkomen.

In beide child classes Game en Publisher maak je, ook boven de klasse definitie zelf, een nieuwe niet-abstracte *public class* **GameRecord** respectievelijk **PublisherRecord**, en baseer beide op de ModelRecord class.

Bedenk waarom de ModelRecord class een 'abstracte' class is, en waarom GameRecord en PublisherRecord 'niet-abstracte' classes zijn!

Maak in de ModelRecord class twee *auto properties*: **Id** van het type **int** en **Name** van het type **string**. Dit zijn properties die beide child classes gemeen hebben.

In de GameRecord class maak je de *auto properties* **Description** van het type **string**, en **Rating** (t.b.v. de *video game content rating*) en **PublisherId** van het type **int**. En in de

PublisherRecord class de properties **Headquarters** en **Website**, beide van het type **string**. Dit zijn properties alleen in de desbetreffende child class nodig zijn.

5.2 Stap 2

We gaan nu de definitie van de dictionary *table_data* aanpassen zodat ze gebruik kunnen gaan maken van de in de vorige stap gemaakte *Record* klassen, om precies te zijn door het type van de *value* van de key/value pair aan te passen.

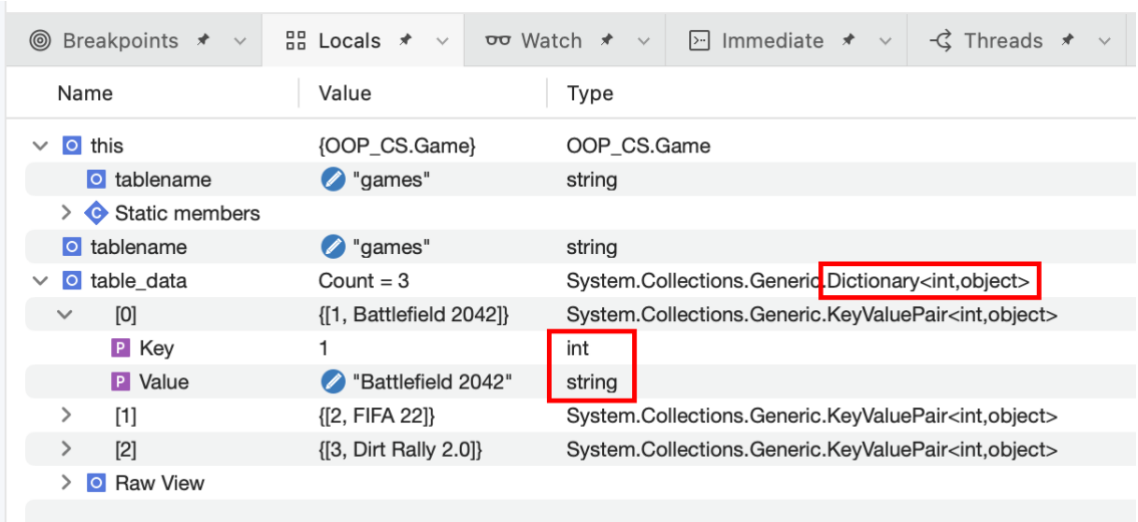
Verander in de Model class de key/value type definitie van de dictionary *table_data* van <int, string> naar <int, **object**>, en pas de overal waar nodig de code aan zodat ze conform de nieuwe definitie zijn.

Doe dezelfde type aanpassing voor de *KeyValuePair* definities in deze class, aangezien deze de items uit de aangepaste dictionary representeren.

Zorg ervoor dat alles nog steeds bouwt.

5.3 Stap 3

Nu is er *run-time* op dit moment nog niet heel veel veranderd. Als je met de debugger door de code loopt zul je zien dat het variabele type **object** nog steeds wordt gebruikt om **string** types in op te slaan. Dit komt omdat we er een string instoppen (bv "Battlefield 2042") op het moment dat we de dictionary vullen:



Maar nu gaan we het type object gebruiken om er Game en Publisher records in op te slaan. Een kwestie dus van het juiste type erin stoppen op het moment dat we records toevoegen aan de dictionary.

Verander in de constructor van de Game class, op de plek waar je de dictionary vult met key/value pairs, de value parameter van de *Add()* method zodat er een **nieuw** object van de class **GameRecord** wordt opgeslagen. Gebruik hiervoor de auto properties binnen deze klasse om de verschillende record velden een waarde te geven. Let wel: dit kan (nog steeds) in 1 statement! (eventueel t.b.v. de leesbaarheid opgedeeld in 2+ regels)

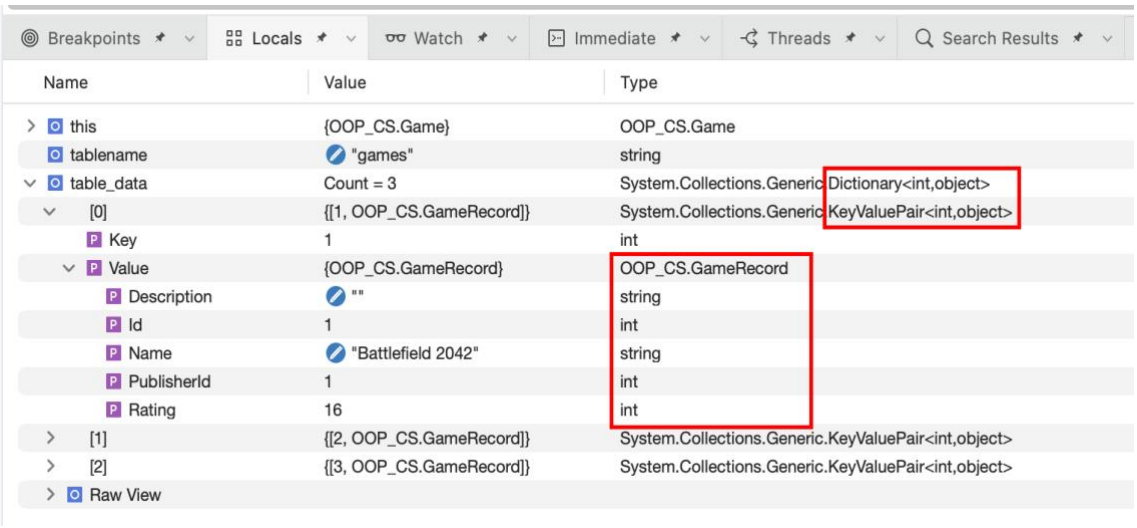
Doe deze aanpassing voor alle *Add()* calls in constructor, en doe daarna hetzelfde in de Publisher class d.m.v. het maken van nieuwe **PublisherRecord** objecten.

De waardes voor *Description* (game) en *Website* (publisher) mag je voor de overzichtelijkheid leeg laten, maar dat hoeft natuurlijk niet. Gebruik voor de *Rating* (game) waardes een bestaand *video game content rating system* (zie Google), of een zelfbedachte (eventueel gebaseerd op een bestaande), maar beperk je tot getallen vanwege het integer type van de property, bv "16" voor 16+ etc.

Kies voor de games die je had gekozen de juiste, d.w.z. bijbehorende, publisher id. Zoals bijvoorbeeld onderstaande combinatie:

```
Id = 1, Name = "Battlefield 2042", PublisherId = 1 // GameRecord
Id = 1, Name = "Electronic Arts", <etc> // PublisherRecord
```

Zorg dat alles nog steeds goed bouwt en kijk met de debugger hoe je dictionary er run-time uitziet. Dit zou zoiets als onderstaand moeten zijn:



Zoals je ziet wordt het variabele type **object** nu gebruikt om **GameRecord** class types in op te slaan, wat komt omdat we er nu een GameRecord object instoppen op het moment dat we de dictionary vullen. En voor de PublisherRecord krijg je een soortgelijk resultaat.

5.4 Stap 4

Pas je test code aan. Als het goed is bouwen de Opdracht methods in de Program class nog steeds, maar krijg je niet meer de gewenste *Value* waardes daar waar je de key/value combinaties naar de console print. Dit gebeurt o.a. na de aanroepen van de *find()* en *getTableData()* methods.

Zorg er in eerste instantie voor dat de naam van de game en publisher weer goed wordt weergegeven, en daarna in ieder geval de *publisher id* van elke game (vanwege de relatie tussen game en publisher), en (optioneel) alle andere properties. Dus zoiets als hier onder:

```
> Record found:
[1] 1 | Battlefield 2042 | 1

> Games table:
[1] 1 | Battlefield 2042 | 1
[2] 2 | FIFA 22 | 1
[3] 3 | Dirt Rally 2.0 | 2
```

```
> Record found:
[1] 1 | Electronic Arts

> Publishers table:
[1] 1 | Electronic Arts
[2] 2 | Codemasters
[3] 3 | Nintendo
```

Let erop dat voor elk GameRecord uit de lijst met games, het *Value* deel van de key/value pair nu een object is (lees: van het type object), en dat je dus een *type cast* nodig hebt om de individuele properties te benaderen. De compiler kan namelijk niet *compile-time* raden of er sprake is van een Game of Publisher object.

Het eenvoudigst is om in de foreach loop voor elke Game het *Value* deel eerst in een aparte lokale variabele van het type GameRecord op te slaan (incl. de type cast), alvorens de individuele properties te benaderen en naar de console te printen. Hetzelfde idee voor de publisher records.

Indien je test succesvol is verlopen, *commit* en *push* je de code met als message: **Opdracht 5a**.

5.5 Stap 5

Maak een *full property* genaamd **Name** om de naam van een Game of Publisher record te lezen of te overschrijven. Omdat beide een *name* column in de tabel hebben, is het dus het beste om deze property in de parent Model class te implementeren (niet in de ModelRecord class).

Let wel dat je ook hier een *type cast* nodig hebt, ditmaal naar het *ModelRecord* type, op een soortgelijke manier als je het al hebt gedaan in je test code na de aanroepen van de *find()* en *getTableData()* methods.

In de *get{}* van de *Name* property haal je eerst de waarde op van het *Value* deel van de **record** field (want dat is een *KeyValuePair*), en na de bovengenoemde cast heb je toegang tot de Name auto property die onderdeel is van ModelRecord.

Doe in de *get{}* ook een *null* waarde check voordat je de waarde van *Name* retourneert, en retourneer een lege string als de *Name* gelijk is aan null. Dat kan met een *if-then-else clause*,

maar ook in 1 statement door gebruik te maken van een zogenaamde *null-coalescing operator* (zie Google!).

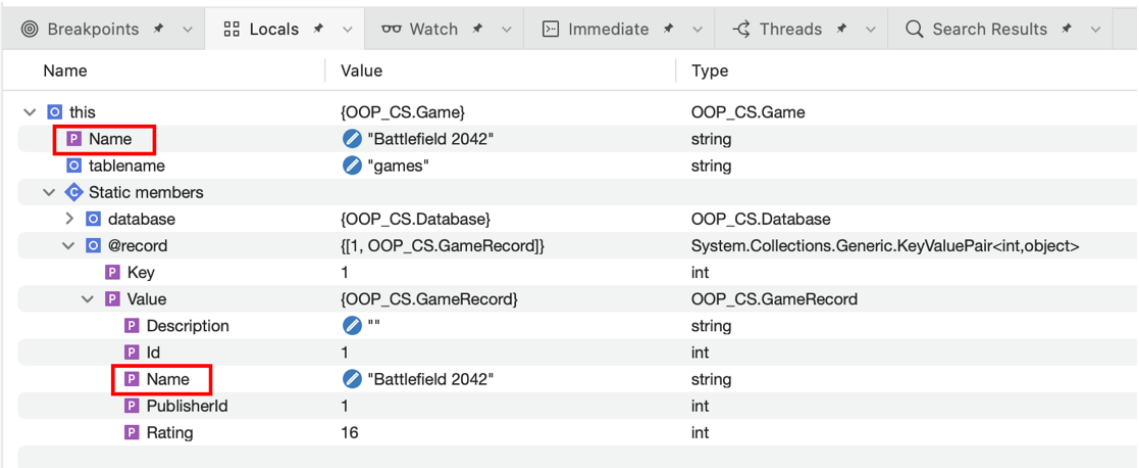
En in de *set{}* doe je het tegenovergestelde, en gebruik makend van het keyword **value**. Het is netjes om (uiteindelijk) zowel de get als set actie in 1 statement te implementeren, want dat kan.

5.6 Stap 6

Test de *Name* property door in het hoofdprogramma een nieuwe method **Opdracht5** te maken en daarin nieuwe instanties van de Game en Publisher classes te maken. Gebruik daarna de property om de naam van een game en publisher te tonen (get), en te overschrijven (set). Toon m.b.v. een print statement de naam van het game en publisher zowel voor als na de set actie.

Vergeet niet om voor het opvragen een *find()* actie te doen zodat het *record* field in het Model object ook daadwerkelijk de record data bevat van de gevonden game of publisher.

Loop ook met de debugger stap voor stap door de code van de property, en bekijk hoe de get en set statements van zowel de Name *full property* (in de Model class) als ook de Name *auto property* (in de ModelRecord class) worden doorlopen:



Controleer of de print statements de verwachte output genereren, dus zoiets als:

```
> Game name (before): Battlefield 2042
> Publisher name (before): Electronic Arts

> Game name (after): Battlefield 2042 updated
> Publisher name (after): Electronic Arts updated
```

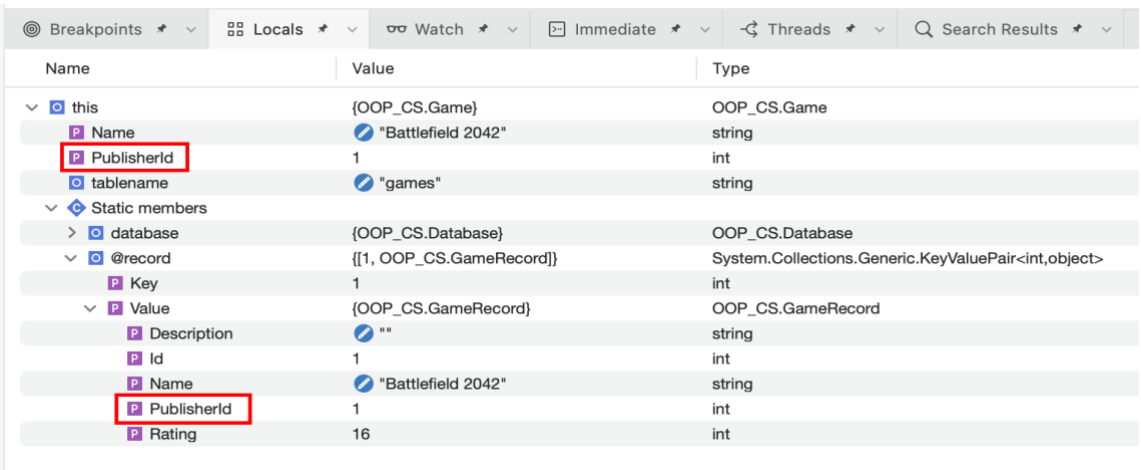
Indien je test succesvol is verlopen, *commit* en *push* je de code met als message: **Opdracht 5b**.

5.7 Stap 7

Maak een *full property* genaamd **PublisherId** om de *publisher id* van een Game record te lezen of te overschrijven. Bedenk zelf in welke class je deze property moet implementeren en waarom.

Test de property door je bestaande test code in de *Opdracht5()* method uit te breiden door de property te gebruiken om de *publisher id* van een game te tonen en te overschrijven.

Analyseer de werking van de property door weer met de debugger stap voor stap door de code van de property te lopen: vanaf de test method door de verschillende classes en weer terug.



Controleer of de print statements de verwachte output genereren, dus zoiets als:

```
> Game name (before): Battlefield 2042
> Publisher name (before): Electronic Arts

> Game name (after): Battlefield 2042 updated
> Publisher name (after): Electronic Arts updated

> Game publisher id (before): 1
> Game publisher id (after): 101
```

Indien je test succesvol is verlopen, *commit* en *push* je de code met als message: **Opdracht 5c**.

5.8 Stap 8 [Optioneel]

We hebben nu voldoende geoefend met het (in C# vaak gebruikte) type *Dictionary* en ook met *KeyValuePair*. Voor het unieke *id* veld in de game en publisher tabellen wordt op dit moment de (tevens unieke) *key* deel van beide data types gebruikt. Maar omdat deze identificatie nu ook door de *Id* property in de Record classes wordt gerepresenteerd, hebben we nu dubbele informatie in onze datastructuur.

Dit is natuurlijk altijd een ongewenste situatie, dus verander het *Dictionary<int, object>* type van de *table_data* property in een **List<object>** type, waardoor de key overbodig wordt en de *Id* property als identificatie kan worden gebruikt.

Deze verandering heeft op veel bestaande properties en methods invloed, dus loop 1 voor 1 de foutmeldingen na en kies de volgens jou correcte aanpassing.

Test je code aanpassingen, zorg dat je hetzelfde testresultaat als voorheen krijgt (een zogenaamde *regressie test*, https://nl.wikipedia.org/wiki/Regressietest), en indien dat het geval is *commit* en *push* je de code met als message: **Opdracht 5d**.

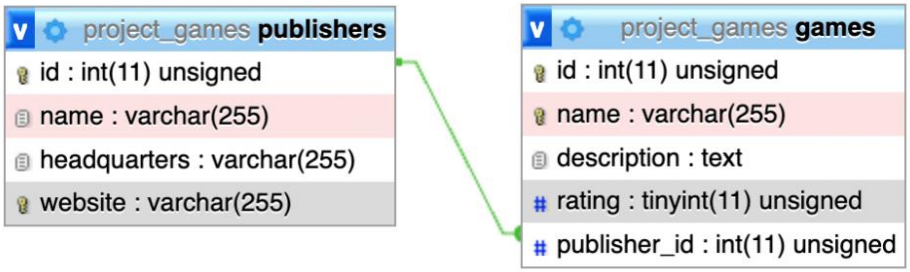
6 Opdracht 6 - Relations… (need I say more?)

Je gaat in deze opdracht een **relatiemethod** toevoegen aan de Game class. Lees voor deze opdracht hoofdstuk 5 'Method chaining en cascading' uit de reader. Lees ook onderstaande inleiding voor je met coderen begint. Dan begrijp je beter waarom je het zo aan moet pakken als in de stappen is beschreven.

Er bestaat een relatie tussen de games en de publishers:

- Een publisher (b.v. Electronic Arts) kan meerdere games hebben uitgebracht (b.v. Battlefield 2042, FIFA 22, etc.).
- En andersom: een game (b.v. Dirt Rally) is uitgebracht door (hooguit) één publisher (oftewel: Codemasters).

Je kunt dit ook terug zien in het zogenaamde *Entity Relationship Model* (ERD, https://nl.wikipedia.org/wiki/Entity-relationshipmodel) van de database, waar we later in meer detail op in zullen gaan als we de gegevens uit een echte database halen. Maar ook zonder database-kennis zou je inmiddels het onderstaande plaatje moeten kunnen begrijpen, of op zijn minst de velden moeten kunnen herkennen. Je ziet de bovenstaand beschreven relaties terug in het lijntje tussen de twee tabelvelden.



Je wilt daarom ook van elk game de naam van de publisher kunnen opvragen en tonen aan de gebruiker. En het streven is om dit d.m.v. 1 statement te kunnen uitvoeren:

```
Game game = new Game("games");
game.find(2);
var game_publisher_name = game.publisher().Name;
```

Aan de laatste regel kun je zien dat *game.publisher()* een object moet zijn, en wel een object van de class Publisher. Dat is de reden dat je de naam van de publisher als property kunt opvragen.

De naam is immers een databaseveld van de tabel publisher, en de Name property zorgt ervoor dat de name uit het veld *publisher.record* wordt gelezen.

Hoe zorg je ervoor dat de juiste publisher uit de database wordt gelezen? Je zou het vooraf kunnen doen, met een expliciete opdracht, maar dat is niet nodig. Je gaat het zó coderen dat de publisher pas uit de tabellen wordt gelezen, op het moment dat de publisher opgevraagd wordt.

6.1 Stap 1

Implementeer in de Game klasse de public method **publisher()**. Die method moet het juiste publisher object retourneren, d.w.z. de bij het huidige game object behorende publisher object.

Maak hiervoor gebruik van de in de vorige opdracht gemaakte Game property *PublisherId*, die de verwijzing naar de publisher van de game bevat. Met de waarde van deze id kun je vervolgens een *find()* actie doen op een nieuw aangemaakte Publisher instantie.

De aanroep van de *find()* method, met de *publisher id* van de game als input parameter, zorgt ervoor de *record* property van het Publisher object de juiste waarde krijgt. Je hoeft daarom niets met de return value van de find() method te doen, die is niet interessant in dit geval, maar laat de publisher() method het Publisher object retourneren.

Als je code nog steeds goed bouwt, *commit* en *push* je de code met als message: **Opdracht 6a**.

6.2 Stap 2

Test of je method werkt in het hoofdprogramma. In een nieuwe test method **Opdracht6()** vraag voor een aantal games de naam van de publisher op d.m.v. 1 statement (zie uitleg bovenin dit hoofdstuk) en toon deze aan de gebruiker.

Je maakt dus een nieuwe game instantie aan en doet er een *find()* actie op m.b.v. het id van de game waarvan je de publisher wilt opvragen, zodat de *record* property van de game de gewenste informatie bevat. Je controleert m.b.v. de return waarde of de game bestaat en als dat het geval is vraag je de naam van de publisher op, dat laatste dus d.m.v. 1 statement.

Dit zou in de console moeten resulteren in zoets als:

```
> Game Battlefield 2042 is published by Electronic Arts
```

Loop ook wederom met de debugger door de code van je nieuwe *publisher()* method, en analyseer of de lokale variabelen de correcte inhoud hebben. Je kunt hier ook goed de relatie zien die je hebt gelegd tussen de *PublisherId* van de game, en de *Id* van de publisher. Onder een screenshot met de inhoud van het Game object, vlak voor het verlaten van de *Game::publisher()* method, en dus vlak na het vinden van het bijbehorende publisher object.

Locals | Search Results

Name	Value	Type
this	{OOP_CS.Game}	OOP_CS.Game
Name	"Battlefield 2042"	string
PublisherId	1	int
database	{OOP_CS.Database}	OOP_CS.Database
@record	{[1, OOP_CS.GameRecord]}	System.Collections.Generic.KeyValuePair<int,object>
tablename	"games"	string
Static members		
id	1	int
name	"Battlefield 2042"	string
publisher_id	1	int
publisher	{OOP_CS.Publisher}	OOP_CS.Publisher
Name	"Electronic Arts"	string
database	{OOP_CS.Database}	OOP_CS.Database
@record	{[1, OOP_CS.PublisherRecord]}	System.Collections.Generic.KeyValuePair<int,object>
Key	1	int
Value	{OOP_CS.PublisherRecord}	OOP_CS.PublisherRecord
Headquarters	"USA"	string
Id	1	int
Name	"Electronic Arts"	string
Website	""	string
tablename	"publishers"	string
Static members		

Geef ook een foutmelding als de game niet wordt gevonden (oftewel test ook met ongeldige game id's).

Indien je test succesvol is verlopen, *commit* en *push* je de code met als message: **Opdracht 6b**.

6.3 Stap 3 [Optioneel]

Zorg ervoor dat je met onderstaande statement ook de naam van een publisher kunt ophalen, oftewel zonder de haakjes () achter het publisher object:

```
game_publisher_name = game.publisher.Name;
```

Geen gedetailleerde aanwijzingen ditmaal, alleen de aanwijzing dat je een nieuwe property moet maken die het publisher object teruggeeft.

Test je aanpassing en indien succesvol verlopen, *commit* en *push* je de code met als message: **Opdracht 6c**.

7 Opdracht 7 - Yet another relation

Bestudeer voor deze opdracht hoofdstuk 5 van de reader (maar dat had je al gedaan voor de vorige opdracht).

In deze opdracht ga je de omgekeerde relatiemethod maken van de vorige opdracht. Daar heb je de method `game.publisher()` geschreven: een game heeft (over het algemeen) één publisher die de game heeft uitgebracht. Omgekeerd heb je de method `publisher.games()` nodig: een publisher heeft (over het algemeen) meerdere games uitgebracht.

De method *Game::publisher()* retourneert een object van de class Publisher. De method *Publisher::games()* moet alle bij een publisher-object horende games retourneren als lijst van objecten van de class Game. Hiervoor wordt de *PublisherId* property van de games vergeleken met de *Id* property van de publisher.

7.1 Stap 1

Voordat we onze tweede relatiemethod gaan implementeren, doen we eerst een voorbereidende stap. Op dit moment krijgen we een dictionary met data terug als we alle gegevens uit een tabel willen ophalen. Dit type is echter niet geschikt om *method chaining* (zie de reader) op toe te passen.

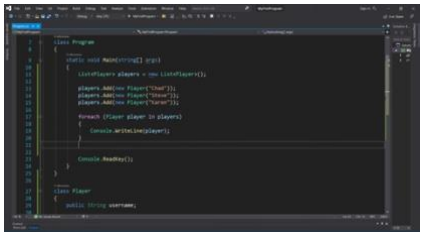
We gaan daarom een method toevoegen die een lijst van objecten, gerelateerd aan de inhoud van de desbetreffende databasetabel, kan teruggeven. De naam van de method wordt **getDataObjects**. Deze method heeft geen parameters en komt in de class Game. De method kan als volgt worden gebruikt:

```
var games = game.getDataObjects();
```

Als er geen records zijn, levert de method een lege lijst op. Als er wel records zijn, levert de method een lijst van class objecten op. Dus in bovenstaand voorbeeld is games een lijst van objecten van de class Game. Gebruik daarom als return type een **List** van het type **Game**.

Zie ook onderstaande link en video over het maken van een *List of objects*:

https://www.c-sharpcorner.com/UploadFile/mahesh/create-a-list-of-objects-in-C-Sharp



Klik op het plaatje om de video tutorial te bekijken. Of als je dit document in een browser bekijkt gebruik dan deze link: https://www.youtube.com/watch?v=aLhAmimoQj8

Het stappenplan voor de Game class:

- Maak de public method **getDataObjects**, zoals gezegd *return type* **List** of type **Game**.
- Initialiseer een lege lijst genaamd **games_objects**, ook een **List** of type **Game**. Retourneer deze lokale variable alvast aan het eind van de method, zodat de code nog steeds bouwt.
- Haal alle table records op m.b.v. getTableData() en stop het resultaat in een variabele, b.v. **games_records**.
- Loop door de records (KeyValuePairs) van de table (dictionary), op dezelfde manier zoals je b.v. al in de test code hebt gedaan. Maak voor elk record een **nieuw** (leeg) Game object aan, b.v. met de naam **game_obj**.
- Gebruik de setRecord() method om de inhoud van het huidige record in het nieuwe object te schrijven, en voeg daarna het object toe aan de games list.
- Retourneer de games list aan het eind van de method.

Test je code. Maak in hoofdprogramma een nieuwe test method **Opdracht7()** met daarin een nieuwe instantie van de Game class, en roep hierop de nieuwe method getDataObjects() aan. Loop met de debugger stap voor stap door de code van de method, en analyseer of je Game objecten met de goede waardes gevuld worden.

Toon in de test method m.b.v. een foreach loop de naam van elke game. Hier kun je dus gebruik maken van de *Name* property, aangezien die direct is gelinked aan de inhoud van

het record veld die je in de nieuwe method had toegekend voor elke game. Dit zou in zoiets als onderstaand moeten resulteren:

```
> Games (from object list):
Battlefield 2042
FIFA 22
Dirt Rally 2.0
```

Indien je test succesvol is verlopen, *commit* en *push* je de code met als message: **Opdracht 7a**.

7.2 Stap 2

Schrijf in de class **Publisher** de public method **games()**. Deze method moet de juiste game-objecten retourneren in de vorm van een lijst. Gebruik daarom als *return type* een **List** van het type **Game**.

In de nieuwe method begin je daarom met het declareren van een lokale variable genaamd **publisher_games** (de games behorende bij een bepaalde publisher) van het bovengenoemde lijst type. Retourneer deze lokale variable alvast aan het eind van de method, zodat de code nog steeds bouwt.

7.3 Stap 3

Schrijf een *full property* om de id van een Game of Publisher record te lezen en te overschrijven. Omdat beide een *id* column in de tabel hebben, is het dus het beste om deze property in de parent Model class te implementeren (niet in de ModelRecord klasse). Geef de property de naam **Id**.

Doe in de *get{}* eerst een null waarde check op het *Value* deel van de *record* field voordat je de waarde van de *Id* opvraagt en retourneert. Dat voorkomt een *NullPointerException* als je later in je test code records met ongeldige id's gaat opvragen. Retourneer de waarde -1 (het *return type* is int) of 'default' (waarde 0 voor een int) in geval van een null waarde.

De *set{}* moet *private* worden, een *id* mag niet van buitenaf aanpasbaar zijn.

7.4 Stap 4

Om te kunnen zoeken naar de 'juiste' games beginnen we in de games() method met het ophalen van alle games. Gebruik hiervoor de Game method *getDataObjects()* en voer deze uit op een nieuw, lokaal aangemaakt, game object. Deze method roep je al aan in je test code, dus hergebruik deze werkende code!

Je hebt nu een lijst met alle game objecten.

7.5 Stap 5

Vervolgens loop je met een *foreach loop* over de lijst met game objecten en filter je de juiste eruit, oftewel de games die een *publisher id* hebben die gelijk is aan de *id* van de huidige publisher (de class waarin we nu zitten). Maak hiervoor gebruik van de in de vorige stap gemaakte Publisher property *Id* (oftewel this.Id), en vergelijk die met de property *PublisherId* van elke game uit de lijst.

Indien ze gelijk zijn voeg je de game toe aan de lijst met 'juiste' games, geïmplementeerd in de lokale variable *publisher_games*. Zorg, indien nog niet gedaan, dat de method de waarde van deze lokale variable retourneert.

Indien je project nog steeds goud bouwt, *commit* en *push* je de code met als message: **Opdracht 7b**.

7.6 Stap 6

Test de games() method door een aanroep in het hoofdprogramma in de reeds gemaakte test method Opdracht7(). Maak hierin een Publisher object aan en doe daarop een find actie met een geldige *publisher id*, zodat het *record* veld (en dus ook de Publisher *Id*) een geldige waarde krijgt. Daarna voer je de nieuwe method uit op het publisher object:

```
publisher.games()
```

Print een lijst met de namen en id's van de gevonden games door gebruikt te maken van een *foreach loop* op de lijst met games. Hiervoor kun je dus gebruik maken van de full properties *Name* en *PublisherId* van de game. Dit zou in zoiets als dit moeten resulteren:

```
> Games belonging to PublisherId 1:
Battlefield 2042 | 1
FIFA 22 | 1
```

Voer dezelfde test ook uit met ongeldige publisher id's. Indien dat het geval is print dan een foutmelding, en als een bepaalde publisher geen games heeft gemaakt print dan een message zoals 'No games found' of iets dergelijks.

Indien je test succesvol is verlopen, *commit* en *push* je de code met als message: **Opdracht 7c**.

7.7 Stap 7 [Optioneel]

Identificeer in de method games() de juiste game met behulp van LINQ (Language Integrated Query). Gebruik hiervoor de *Method Based Syntax*, want die sluit het best aan bij de OOP manier van programmeren. Zie ook onderstaande links met uitleg over LINQ:

https://www.geeksforgeeks.org/linq-method-syntax

Klik op het plaatje om de video tutorial te bekijken. Of als je dit document in een browser bekijkt gebruik dan deze link: https://www.youtube.com/watch?v=X5UW3BIx9fE

Herhaal de in deze opdracht gemaakte testen, en indien succesvol verlopen, *commit* en *push* je de code met als message: **Opdracht 7d**.

7.8 Stap 8 [Optioneel]

Als je de vorige (optionele) stap hebt geïmplementeerd heb je gemerkt dat je heel eenvoudig met 1 query statement op zoek kan gaan naar een bepaalde record in een tabel. Dat nodigt uit om een method te maken waarin je ook kunt zoeken op bijvoorbeeld de naam van een game of publisher.

Hernoem de method *find()* naar **findById()** (d.m.v. een *refactor > rename* actie). Maak in de Model class een nieuwe method **findByName()** met een string type als input parameter die het juiste record met de gezochte naam terug geeft. Maak voor beide methods gebruikt van LINQ.

Test de nieuwe method en indien succesvol verlopen, *commit* en *push* je de code met als message: **Opdracht 7e**.

8 Opdracht 8a - The real deal (database interaction) – Part 1 Setup

In de volgende twee opdrachten gaan we alle tot nu toe gesimuleerde database acties (d.m.v. dictionaries en KeyValuePairs) vervangen door echte database acties. Hiervoor ga je op je eigen laptop, de zogenaamde *local host*, een MySQL database configureren. Nadat we de database en tabellen hebben gemaakt en gevuld hebben met test gegevens, gaan we deze uit de database lezen en in de reeds gemaakte objecten opslaan.

Omdat dit de eerste keer is dat je tijdens deze studie met een database gaat communiceren, gaan we deze stappen klassikaal en gezamenlijk doen. Daarom zijn de stappen niet allemaal in detail uitgewerkt zoals je dat van de vorige opdrachten gewend bent, maar worden ze over het algemeen globaal beschreven.

8.1 Stap 1

Installeer MAMP (https://nl.wikipedia.org/wiki/MAMP) voor de MacBook gebruikers, of XAMPP (https://nl.wikipedia.org/wiki/XAMPP) voor de Windows gebruikers. Deze bevat o.a. een Apache webserver en een MySQL databaseserver:

https://www.mamp.info/en/downloads (MAMP voor Mac)
https://www.apachefriends.org/download.html (XAMPP voor Windows)

Instructies voor Windows:
https://www.jcchouinard.com/install-mysql-and-phpmyadmin-with-xampp

Klik op het plaatje om de video tutorial te bekijken. Of als je dit document in een browser bekijkt gebruik dan deze link: https://www.youtube.com/watch?v=ahupFE78tVc

Als alternatief mag je ook MySQL i.c.m. MySQLWorkbench installeren, dus zonder webserver.

https://dev.mysql.com/downloads (zowel de 'Community Server' als de 'Workbench')

8.2 Stap 2

PhpMyAdmin is een MySQL client in de vorm van een web applicatie op je eigen lokale Apache server, de *local host*, die als onderdeel van MAMP/XAMPP is geïnstalleerd.

Doe een local host en phpMyAdmin check:

http://localhost:8888/phpMyAdmin

Doe, indien van toepassing, een MySQLWorkbench check.

8.3 Stap 3

Maak een database aan: **project_games** *[in PhpMyAdmin: Home > Databases > Create, of de New knop; GEEN spaties of andere rare tekens]*

Maak in de database de volgende tabellen aan: **games** en **publishers** *[Database project_games > Structure > Create table; GEEN spaties of andere rare tekens]*

Maak in de *games* tabel de onderstaande velden aan. Bij de niet genoemde velden vul je niets in c.q. maak je geen keuze. Kies Go als je een pop-up schermpje krijgt.

- **id** - type: int, length: 11, attributes: unsigned, index: primary, A_I: aangevinkt (auto-increment).
- **name** - type: varchar, length: 255, index: unique.
- **description** - type: text.
- **rating -** type: tinyint, length: 11, attributes: unsigned.

Druk op de Save knop. Je komt nu automatisch in het *Table Structure* scherm:

Server: localhost:8889 » Database: project_games » Table: games

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)		UNSIGNED	No	*None*		AUTO_INCREMENT	Change Drop More
2	name	varchar(255)	utf8_general_ci		No	*None*			Change Drop More
3	description	text	utf8_general_ci		No	*None*			Change Drop More
4	rating	tinyint(11)		UNSIGNED	No	*None*			Change Drop More

Maak in de *publishers* tabel de onderstaande velden aan. Bij de niet genoemde velden vul je niets in c.q. maak je geen keuze. Kies Go als je een pop-up schermpje krijgt.

- De velden **id** en **name** die ook in de *games* tabel staan, beide met dezelfde structuur als in de *games* tabel.
- **headquarters** - type: varchar, length: 255 (dus index niet unique)
- **website**, dezelfde structuur als het *name* veld (dus index wel unique)

Druk op de Save knop. Je komt nu automatisch in het *Table Structure* scherm:

Server: localhost:8889 » Database: project_games » Table: publishers

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)		UNSIGNED	No	*None*		AUTO_INCREMENT	Change Drop More
2	name	varchar(255)	utf8_general_ci		No	*None*			Change Drop More
3	headquarters	varchar(255)	utf8_general_ci		No	*None*			Change Drop More
4	website	varchar(255)	utf8_general_ci		No	*None*			Change Drop More

8.4 Stap 4

Maak een gebruiker **noorderpoort** aan, wachtwoord **1234**, of een ander wachtwoord natuurlijk, maar sla hem dan ergens veilig op! *[Home > User accounts > Add user account; Host name: localhost; GEEN spaties of andere rare tekens; GEEN 'Global privileges' toewijzen]*

Geef de *noorderpoort* gebruiker toegang tot de *project_games* database *[Home > User accounts > Edit privileges; Database knop, selecteer database in dropdown, Go; selecteer 'Database-specific privileges', Go knop]*

8.5 Stap 5

Vul de tabellen met test data. Je kunt dezelfde data gebruiken die je in de voorgaande hoofdstukken al hebt gebruikt voor de dictionaries.

8.6 Stap 6

We gaan een *foreign key* veld definiëren in de *games* tabel die de relatie tussen elke game en de desbetreffende publisher bevat. De waarde in het *foreign key* veld is de verwijzing naar het *primairy key* veld **id** in de *publishers* tabel.

Voeg in de *games* tabel het volgende veld toe. Bij de niet genoemde velden vul je niets in c.q. maak je geen keuze *[Table games > Structure > Add 1 column > Go button; GEEN spaties of andere rare tekens]*

publisher_id - type: int, length: 11, default: null, attributes: unsigned.

Druk op de Save knop. Je komt weer terug in het *Table Structure* scherm:

Leg de relatie tussen de foreign key *publisher_id* in de *games* tabel en de primairy key veld *id* in de *publishers* tabel *[Table games > Structure > Relation view]* [1]

- On Delete: Set null; On Update: Cascade
- Column: publisher_id
- Database: project_games, Table: publishers, Column: id

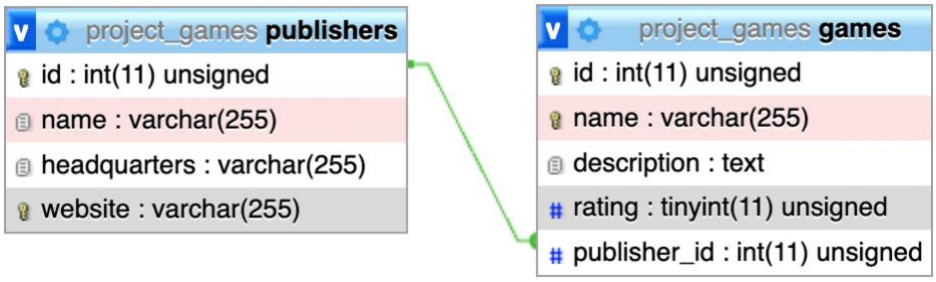
[1] Als het tabblad 'Relation view' niet zichtbaar is dan was mogelijk niet de (default) storage engine InnoDB geselecteerd bij het aanmaken van de tabel. Dit kun je herstellen in het *Table games > Operations* scherm.

Druk op de Save knop. Je komt weer terug in het *Table Structure Relation view* scherm.

8.7 Stap 7

Controleer de structuur van de tabellen en de relatie ertussen in de designer *[Database project_games > Designer]*

Als je alle stappen goed hebt uitgevoerd zou je het onderstaande resultaat moeten zien:



Je ziet de één-op-veel-relatie tussen de publishers (de één-kant) en de games (de veel-kant) terug in het groene lijntje:

- Een publisher (b.v. EA) kan meerdere games hebben uitgebracht (bv Battlefield 2042, FIFA 22).
- En andersom: een game (b.v. Dirt Rally 2.0) is uitgebracht door (hooguit) één publisher (oftewel: Codemasters).

8.8 Stap 8

Je hebt nu best veel stappen gedaan (lees: veel tijd besteed) om de games database te ontwerpen en te vullen met waardevolle test data. Het zou daarom zonde zijn als door een onbedoelde fout tijdens het toevoegen of deleten van gegevens de database ineens onbruikbaar wordt en je (mogelijk) helemaal opnieuw moet beginnen.

Daarom kun je het ontwerp en de gegevens in je database *exporteren* naar een file: een SQL tekst bestand. Eenmaal geëxporteerd kun je met behulp van deze file 'met een druk op de knop' de database inclusief de gegevens herstellen d.m.v. een *import* actie.

Exporteer de games database naar een file genaamd **project_games.sql** *[Database project_games > Export]*. Voeg deze toe aan je solution, bekijk de inhoud van de file en kijk

of je de stappen die je in dit hoofdstuk handmatig hebt gedaan herkent in de SQL statements.

Hierna *commit* en *push* je de code met als message: **Opdracht 8.1**.

8.9 Stap 9

Nu we een echte database hebben kunnen we in onze C# applicatie de game gegevens uit de database lezen en vervolgens opslaan in onze bestaande datastructuur: de dictionaries. Hiervoor gaan de singleton class **Database** gebruiken waarmee we al een start hebben gemaakt en waarin we al een *connection* property van het type *SqlConnection* hebben gedefinieerd. Dat doen we in het volgende hoofdstuk.

Opdracht 8b - The real deal – Part 2 Implementation

Bestudeer voor deze opdracht de tekst in onderstaande link:
https://www.guru99.com/c-sharp-access-database.html

Lees goed hoe een *SQL connection string* wordt opgebouwd, hier gaan we in dit hoofdstuk zodadelijk mee beginnen.

8.10 Stap 1

Zoals je hierboven hebt gelezen is de eerste stap om data uit een database te kunnen lezen het opzetten van een connectie met de database. Wat we inmiddels al hebben is een vrijwel lege *Database* class met daarin een *connection* property van het type *SqlConnection*.

Echter, dit data type is bedoeld om met een (Microsoft) SQL Server te communiceren. En omdat je in het vorige hoofdstuk een (Oracle) MySQL server hebt geïnstalleerd middels XAMPP (of MAMP voor Mac) doen we eerst een kleine herstelactie.

Installeer m.b.v. de *NuGet package manager* de library **MySql.Data** (i.p.v. System.Data.SqlClient) en voeg een *using statement* toe voor de **MySql.Data.MySqlClient** namespace. Verander daarna het type van de *connection* property naar **MySqlConnection** (i.p.v. SqlConnection).

Controleer of je applicatie nog goed bouwt voordat je verder gaat.

8.11 Stap 2

Onze eerste stap in de code wordt dus om de *connection* property de juiste waarde te geven. Definieer in de constructor van de *Database* class een local variable genaamd **connectionString** van het type string en initialiseer deze met de waarde **@""** (incl. het @ teken dus, als je wilt weten waarom: zie Google).

Daarna voeg je aan deze connection string, in de onderstaande volgorde, steeds d.m.v. *string concatenation* (wederom: zie Google) de genoemde hard coded string toe gevolgd door de bijbehorende waarde uit de config file. De hard coded strings met hun uitleg kun je

ook vinden in de bovenstaande link die je hebt bestudeerd, en om de config items op te halen kun je dus de bestaande Config class method *item()* gebruiken:

- `"Data Source="` gevolgd door de waarde van config item "host".
- `";Port="` gevolgd door de waarde van config item "port".
- `";Initial Catalog="` gevolgd door de waarde van config item "database".
- `";User ID="` gevolgd door de waarde van config item "username".
- `";Password="` gevolgd door de waarde van config item "password".

Tenslotte ken je de waarde van de opgebouwde lokale variable *connectionString* toe aan de MySqlConnection class property *ConnectionString*, die onderdeel is van de Database class property *connection* (van type MySqlConnection). Op deze manier kan die later worden gebruikt als je m.b.v. deze property de database connectie gaat openen.

Toon als test de inhoud van de property d.m.v. *console writeline* om de waarde ervan te kunnen inspecteren als je de applicatie uitvoert. Als de waarde er goed uit lijkt te zien, en of die daadwerkelijk goed is zullen we later ervaren, *commit* en *push* je de code met als message: **Opdracht 8.2**.

8.12 Stap 3

Nu we de connection string stap voor stap hebben opgebouwd en hebben toegekend aan de connection property zijn we toe aan de laatste stap: een poging om daadwerkelijk te connecten met onze database.

Een connectie maken met de database kan met de method *open()* die beschikbaar is binnen de *MySqlConnection* class. Roep deze method aan op de *connection* property en zet de aanroep én de foutafhandeling in een *try-catch* statement. Zie Google als dit statement nieuw voor je is.

[Tip] Gebruik *code snippet* 'try' (2x tab tijdens of na het typen van het woord, of Ctrl+Space voor een lijst).

Toon een print statement als de connectie is gelukt, maar ook als die niet goed gaat (oftewel i.g.v. een exception).

Test of je connectie succesvol tot stand is gekomen als je de applicatie uitvoert. Zo niet, loop dan met de debugger stap voor stap door je nieuwe code en kijk of de connectie string goed wordt samengesteld. Ook krijg je op deze manier meer info over een eventuele exception als die optreedt.

Indien je test succesvol is verlopen, *commit* en *push* je de code met als message: **Opdracht 8.3**.

8.13 Stap 4 - CRUD

Je hebt nu een werkende databaseconnectie gemaakt. De volgende stap is om daar iets mee te doen: databasebewerkingen.

Je kunt de volgende basisbewerkingen op een database onderscheiden:

1. Gegevens toevoegen (Create), met een SQL INSERT-query
2. Gegevens ophalen (Read), met een SELECT-query
3. Gegevens wijzigen (Update), met een UPDATE-query
4. Gegevens verwijderen (Delete), met een DELETE-query

De genoemde vier databasebewerkingen worden samen CRUD genoemd: Create, Read, Update, Delete. Lees de onderstaande uitleg over CRUD operaties in C#, hier gaan we in de volgende stappen mee aan de slag.

https://www.c-sharpcorner.com/article/crud-operations-in-windows-applications-using-c-sharp

Zoals gezegd gaan we alle tot nu toe gesimuleerde database acties (d.m.v. Dictionaries en KeyValuePairs) vervangen door echte database acties.

8.14 Stap 5 - Create

De eerste databasebewerking is de **C**reate, oftewel gegevens toevoegen aan de database. De gesimuleerde database actie die we tot nu toe hebben gedaan om gegevens toe te voegen bestaat uit het toevoegen van records (KeyValuePairs) aan de *table_data* dictionary.

Dit doen we op dit moment in de constructor van de Game en Publisher classes. Dus dit is een goede plek om, gebruik makend van wat we al hebben gedefinieerd, te zorgen dat dezelfde data in onze database terecht komt.

Maak in de Database class een nieuwe public method **create()**, met return type **int**, en met input parameters **table_name** van type string, en **record** van het type object (want we willen namelijk óf een Game record, óf een Publisher record wegschrijven).

Implementeer in deze method de code die nodig is om een enkel record in de databasetabel games of publishers te inserten. Geen detail aanwijzingen ditmaal, gebruik de informatie in de CRUD link boven die je hebt gelezen, en van de code die je tot nu toe al hebt ontwikkeld. De return waarde (van type int) van de method kun je gebruiken om de return waarde van de SQL functie die de query uitvoert terug te geven.

Roep vervolgens, voor elk record die je er al hebt gedefinieerd, de *Database::create()* method aan vanuit de Game en Publisher constructors. Test je nieuwe functionaliteit door de applicatie te bouwen en te runnen. Controleer daarna in de SQL client of de nieuwe records daadwerkelijk zijn toegevoegd.

Indien je test succesvol is verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 8.4**.

8.15 Stap 6 - Read

De volgende databasebewerking is de **R**ead, oftewel gegevens lezen uit de database. De gesimuleerde database actie die we tot nu toe hebben gedaan om gegevens te lezen is de Model::find() method. Deze zoekt naar een record (KeyValuePair) in de *table_data* dictionary en retourneert het gevonden record, of een leeg record indien de record niet is gevonden).

Maak in de Database class een nieuwe public method **read(),** met return type **KeyValuePair<int, object>** (type object, want we willen namelijk óf een Game record, óf een

Publisher record lezen en retourneren), en met input parameters **table_name** van type string en **id** van type int.

Implementeer in deze method de code die nodig is om een enkel record uit de databasetabel games of publishers te lezen, waarbij je zoekt op de *id* die je als parameter meekrijgt. Wederom geen detail aanwijzingen ditmaal, gebruik de informatie in de CRUD link boven die je hebt gelezen, en van de code die je tot nu toe al hebt ontwikkeld.

Roep vervolgens in de Program class, op elke plek waar je de *Model::find()* method aanroept op een Game en Publisher object, de *Database::read(<id>)* method aan en print het resultaat van deze actie op dezelfde manier zoals je dat nu ook al doet. Controleer in de SQL client of de geprinte debug info van het gevonden record overeenkomt met de record contents in de database.

Indien je test succesvol is verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 8.5**.

8.16 Stap 7 - Update

De volgende databasebewerking is de **Update**, oftewel gegevens updaten in de database. De gesimuleerde database actie die we tot nu toe hebben gedaan om gegevens te updaten gebeurt in de Program class op de plek waar we d.m.v. *full properties* de Name en PublisherId van een Game record updaten (en de Name van een Publisher record).

Maak in de Database class een nieuwe public method **update()**, met return type **int**, en met input parameters **table_name** van type string, **name** van het type string en **id** van type int.

Implementeer in deze method de code die nodig is om de naam van een enkel record uit de databasetabel games of publishers te overschrijven, waarbij je zoekt op de *id* die je als parameter meekrijgt. Wederom geen detail aanwijzingen ditmaal, gebruik de informatie in de CRUD link boven die je hebt gelezen, en van de code die je tot nu toe al hebt ontwikkeld. De return waarde (van type int) van de method kun je gebruiken om de return waarde van de SQL functie die de query uitvoert terug te geven.

Roep vervolgens in de Program class, op elke plek waar je de naam van een record van een Game en Publisher object update m.b.v. de *full property*, de *Database::update(<name>, <id>)* method aan en print het resultaat van deze actie op dezelfde manier zoals je dat nu ook al doet. Controleer in de SQL client of de geprinte debug info van het gevonden record overeenkomt met de record contents in de database.

Indien je test succesvol is verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 8.6**.

8.17 Stap 8 - Delete

De volgende databasebewerking is de **Delete**, oftewel gegevens deleten in de database. In het vorige deel hebben we hiervoor geen gesimuleerde database actie gedaan.

Maak in de Database class een nieuwe public method **delete()**, met return type **int**, en met input parameters **table_name** van type string, en **id** van type.

Implementeer in deze method de code die nodig is om een enkel record uit de databasetabel games of publishers te verwijderen, waarbij je zoekt op de *id* die je als parameter meekrijgt. Wederom geen detail aanwijzingen ditmaal, gebruik de informatie in de CRUD link boven die je hebt gelezen, en van de code die je tot nu toe al hebt ontwikkeld.

Roep in een nieuwe test method **Opdracht8()** de *Database::delete(<id>)* method aan en print het resultaat van deze actie naar de console. Maak gebruik van de *Database::read(<id>)* om voor én na de delete actie te checken of het record met de betreffende id bestaat.

Indien je test succesvol is verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 8.7**.

9 Opdracht 9 - Zie Scherp Scherper H9

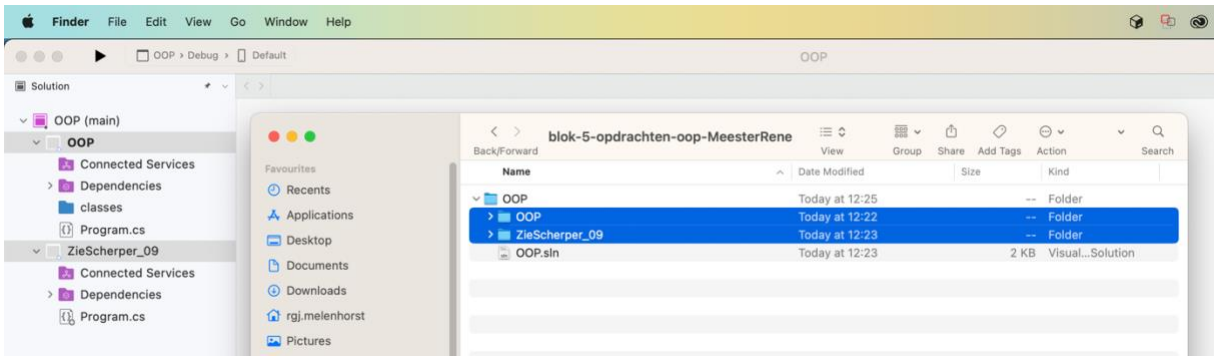
Vanaf dit hoofdstuk gaan we met nieuw lesmateriaal aan de slag en wel uit het boek *Zie Scherp Scherper* van Tim Dams (ISBN: 9789464651560). De theorie uit dit boek staat ook online, dus je hoeft het boek niet aan te schaffen. De oefeningen uit het boek, die niet online staan, zijn d.m.v. screenshots aan dit document toegevoegd.

Hieronder link naar de lesstof, we starten bij *H9 Object Oriented Programming*:
https://apwt.gitbook.io/zie-scherp-scherper

GitHub repo en MS Visual Studio

We blijven werken in de repo waarin we de *Games Database Applicatie* hebben gemaakt, maar we maken voor de overzichtelijkheid vanaf nu wel voor elk nieuw hoofdstuk een nieuw C# project aan.

Dus maak in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_09** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project):



Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_09**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Zorg dat het 'Hello World' print statement in de Main() terecht komt, en bouw en run het project. Als je het print statement in je console verschijnt ga je verder naar de volgende stap.

9.1 Stap 1 - OOP Intro

Maak (in Visual Studio) in je nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

Lees het eerste deel van dit hoofdstuk genaamd *OOP Intro*:
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h9-object-oriented-programming/0_oop_intro

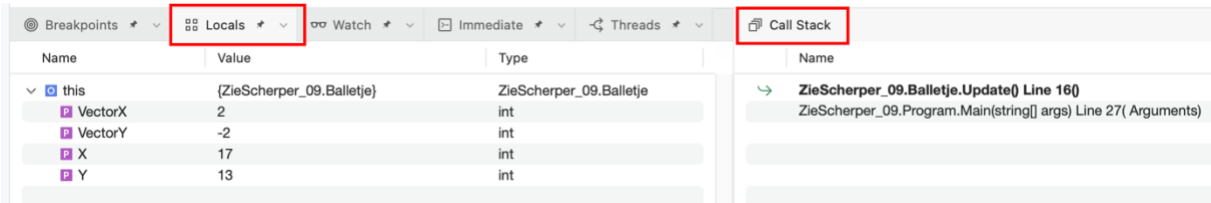
Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Zorg dat in Visual Studio de beschreven code van de class **Balletje**, bij het kopje 'Een wereld mét OOP: Pong', in je nieuwe C# project terecht komt, en wel in de folder **classes**.
2. Dit doe je op dezelfde manier zoals je gewend bent van de vorige opdrachten, en die tevens als tip op de webpagina wordt genoemd:

   > ⓘ De code voor een nieuwe klasse schrijf je best in een apart bestand in je project. Klik bovenaan in de menu balk op "Project" en kies dan "Add class...". Geef het bestand de naam "Balletje.cs".

3. Zorg nu dat de beschreven code van de Main() method, waarmee je o.a. een balletje-object maakt, in de Main() method van je **Program** class terecht komt.
   Het 'Hello World' print statement mag je weghalen of in commentaar zetten.
4. Bouw en run het project. Als je er een bewegend 'balletje' in je console verschijnt ga je verder met het toevoegen van de beschreven code voor twee balletjes, en als dat ook goed werkt voor de 3+ balletjes.

Test je code ook door er met de debugger stap voor stap doorheen te lopen, b.v. door een *breakpoint* te zetten in de method Balletje::Update(). Bekijk tijdens het debuggen de (veranderende) waardes van de gebruikte lokale variabelen, en gebruik de *Call Stack* om te analyseren voor welk balletje in de array de update() method wordt aangeroepen.



Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 9.1**.

9.2 Stap 2 - Klassen en objecten in C#

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht09_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Klassen en objecten in C#*:
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h9-object-oriented-programming/0b_oopincs

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Maak in je huidige Visual Studio C# project een (lege) class **Auto** op de wijze zoals beschreven in de tutorial (kopje 'Klassen in Visual Studio toevoegen'). Plaats de file wederom in de folder **classes**.
   Maak in de Main() method van je **Program** class zoals beschreven objecten van de class Auto (kopje 'De new operator'), dus ook op de manier die een error veroorzaakt.
2. Zet de statements die errors en warnings veroorzaken in commentaar zodat je solution weer goed bouwt. Run daarna het project.

Indien het project goed bouwt en runt, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 9.2**.

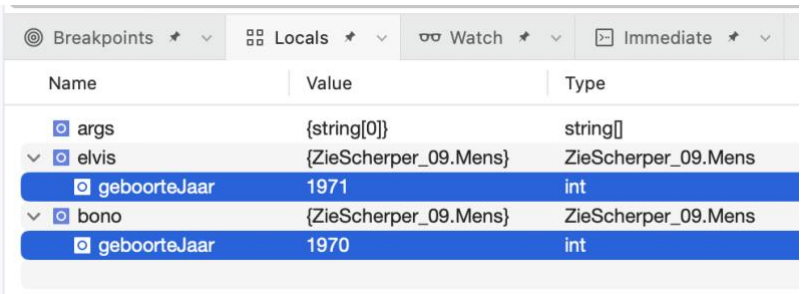
9.3 Stap 3 - Methoden en access modifiers

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht09_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

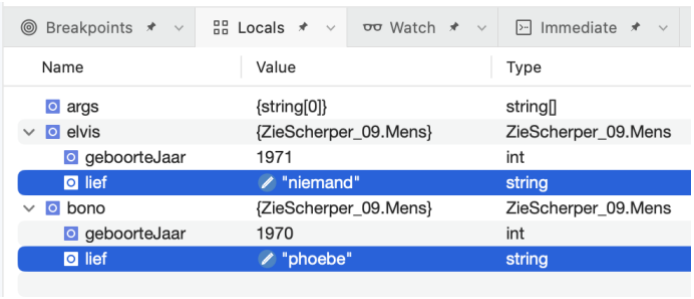
Lees het volgend deel van dit hoofdstuk genaamd *Methoden en access modifiers:* https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h9-object-oriented-programming/0c_simpleobjects

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de genoemde code van de class **Mens** en met daarin method **Praat()** op in je C# project (in een nieuwe class file!), en roep de Praat() method in de Main() method 2 maal aan zoals beschreven (kopje 'Object methoden').
   Run je project en test of je de beschreven output krijgt.
2. Voeg de code van method **VertelGeheim()** toe aan de class Mens en probeer of je deze methode vanuit de Main op één van beide Mens objecten kan aanroepen. Probeer daarna, zoals beschreven, of je dezelfde method vanuit de Mens::Praat() method kunt aanroepen. Run je project en test of je de beschreven output krijgt.
3. Voeg zoals beschreven de instantie variabele (a.k.a. data veld of data field) **geboorteJaar** toe aan de Mens klasse en zorgt dat de waarde geprint wordt in de Praat() method.
4. Voeg de code van method **StartVerjongingskuur()** toe aan de class Mens. Maak zoals beschreven in de Main() method 2 objecten aan van de class Mens en roep op één ervan de verjongingskuur method aan. Run je project en test of je de beschreven output krijgt.
5. **Test je code ook** door er met de debugger stap voor stap doorheen te lopen. Zet een breakpoint in je Main() method bij de aanroep naar StartVerjongingskuur() en stap met de debugger vanaf dat moment stap voor stap door de code. Analyseer terwijl je dat doet wat er met de waarde van het geboortejaar gebeurt voor beide objecten.



6. Neem in de class Mens de code op betreffende de data field *private string lief*, incl. de nieuwe en veranderde methods die hier gebruik van maken. Roep de nieuwe method VeranderLief() vanuit de Main() method aan op één van beide Mens objecten, en wel vóór de aanroep naar de Praat() method (zie het voorbeeld in de tutorial). Run je project en test of je de beschreven output krijgt.
7. Test je code opnieuw door er ook met de debugger stap voor stap doorheen te lopen. Zet een breakpoint in je Main() method bij de aanroep naar VeranderLief() en stap met de debugger vanaf dat moment stap voor stap door de code. Analyseer terwijl je dat doet wat er met de waarde van de *private string lief* property gebeurt voor beide objecten.



De verwachte output in je console zou na alle stappen zoiets moeten zijn als:

```
Ik ben een mens!
Ik ben geboren in 1971.
Ik ben op niemand verliefd.

Ik ben een mens!
Ik ben geboren in 1970.
Ik ben verliefd op phoebe
```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 9.3**.

9.4 Stap 4 - Properties

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht09_3()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Properties:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h9-object-oriented-programming/2_properties

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de genoemde code van de class **SithLord** op in je C# project (in een nieuwe class file!), en probeer zoals beschreven van deze nieuwe class de private property *sithName* in de Main() method naar de console te printen. Bouw je project en controleer of je zoals verwacht een foutmelding krijgt.
   Zet het statement dat voor de foutmelding zorgt in commentaar.
2. Breidt zoals beschreven de SithLord class uit met de *full property* **Energie** en voeg de bijbehorende code om deze property te *getten* en *setten* toe aan je Main() method. Bouw en run je project en test of je de beschreven output krijgt.
3. Bouw zoals beschreven de extra controle in op de setter van de Energie property, voeg een statement aan de Main() method toe waar je de setter aanroept met een ongeldige waarde, en print de waarde van de property (nogmaals) naar de console. Bouw en run je project en test of je de verwachte output krijgt.
4. Geef zoals beschreven de setter van de Energie property de visibility **private** en bouw je project opnieuw. Los de foutmeldingen op door de genoemde public method **ResetLord()** aan de class SithLord toe te voegen en deze in de Main() method aan te roepen (i.p.v. de *private setter*) om de energie waarde van de SithLord objecten te overschrijven.

Zorg dat je in de ResetLord() method wél gebruik maakt van de *private setter* Energie (en **niet** van de instantievariabele energie).
Roep ook nu ResetLord() aan met een ongeldige waarde, en print de waarde van de property (nogmaals) naar de console.

5. Voeg de genoemde *transformatie property* **Email** toe aan de SithLord class waarbij je de bestaande private property **sithName** gebruikt als gedeelte vóór het @ teken uit de samengestelde email string (i.p.v. de - niet bestaande - genoemde *Voornaam*).
   Voeg een constructor toe aan de class SithLord met als input parameter **name** van het type **string**, die je gebruikt om de private property sithName te initialiseren.
   Pas tenslotte de instantiaties van de SithLord class aan door een naam aan de constructor mee te geven, en print het email adres van de SithLord objecten naar de console.
6. Voeg een *auto property* **Geboortejaar** van type **int** toe aan de class SithLord. Voeg aan je Main() method een statement toe die deze *auto property* een waarde geeft print daarna deze waarde naar de console m.b.v. de getter.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als:

```
Vaders energie is 0
Vaders energie is 20
Vaders email is Vader@ziescherp.be
Vaders geboortejaar is 2000
```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 9.4**.

9.5 Stap 5 - DateTime: leren werken met objecten

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht09_4()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *DateTime: leren werken met objecten:*

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h9-object-oriented-programming/datetime

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de beschreven code betreffende de *static property* **DateTime.Now** op in de Main() method van je Program class.
2. Maak van dezelfde **DateTime** class een object aan waarbij je waardes meegeeft aan de constructor. Type in je code, zoals ook getoond, de *dot-operator* achter de object variabele die je hebt bedacht om de lijst met beschikbare methods en properties te zien.
3. Gebruik één van de genoemde Add-methoden om de waarde van je DateTime object aan te passen. Print het resultaat voor én na de aanpassing naar de console.
4. Gebruik één van de genoemde DateTime properties op je DateTime object en print de waarde ervan voor naar de console.
5. Pas de genoemde *static method* **IsLeapYear()** toe op je DateTime object en print het resultaat als volgt naar de console, afhankelijk van het resultaat:
   ```
   Het jaar 2020 van mijn DateTime object is een schrikkeljaar.
   Het jaar 2022 van mijn DateTime object is geen schrikkeljaar.
   ```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 9.5**.

9.6 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Oefening 'RapportModule'.

Class **Rapport**, test method **Opdracht09_6a()**, commit **Opdracht 9.6a**. Verwachte output zoals beschreven in de oefening.

b) Oefening 'Nummers'.

Class **Nummers**, test method **Opdracht09_6b()**, commit **Opdracht 9.6b**. Verwachte output zoals beschreven in de oefening. Test ook of je 'deling door nul' check goed werkt!

c) Oefening 'Persoon'.

Class **Persoon**, test method **Opdracht09_6c()**, commit **Opdracht 9.6c**. Verwachte output in je console zoiets als:

```
De leeftijd van Pietje Puk is 32
```

d) Oefening 'Verjaardag'.

Geen class, test code in Main(); test method **Opdracht09_6d()**, commit **Opdracht 9.6d**. Verwachte output zoals beschreven in de oefening.

e) Oefening 'BankManager*'

Class **Rekening**, test method **Opdracht09_6e()**, commit **Opdracht 9.6e**. Verwachte output in je console zoiets als onder, die de rekening info voor én na de transactie laat zien:

```
Naam:           Pietje
Rekeningnummer: 1234
Staat:          Geldig
Balans:         €1000

Naam:           Klaasje
Rekeningnummer: 5678
Staat:          Geldig
Balans:         €0

Pietje: Hoeveel geld wil je naar Klaasje overschrijven?
10

Naam:           Pietje
Rekeningnummer: 5678
```

```
Staat:          Geldig
Balans:         €990

Naam:           Klaasje
Rekeningnummer: 5678
Staat:          Geldig
Balans:         €10
```

f) [Optioneel] Oefening 'Bibliotheek'.

Class **BibBoek**, test method **Opdracht09_6f()**, commit **Opdracht 9.6f**. Verwachte output in je console zoiets als:

```
De inleverdatum van het boek van Pietje Puk is 1/29 (mm/dd)
De inleverdatum van het boek van Pietje Puk is verlengd
tot 1/30 (mm/dd)
```

Kobo Desktop

SHOP KOBO MY BOOKS EREADER SETUP *Zie Scherp Scher...*

ZIE SCHERP SCHERPER - 2E EDITIE

9.5 Oefeningen

RapportModule

Ontwerp een klasse Rapport die je zal tonen wat je graad is gegeven een bepaald behaald percentage. Het enige dat je aan een Rapport-object moet kunnen geven is het behaalde percentage (int) dat wordt bijgehouden via een auto-property genaamd Percentage. Via een methode PrintGraad kan de behaalde graad weergegeven worden, gebaseerd op dit percentage. Dit zijn de mogelijkheden:

- Minder dan 50%: "Niet geslaagd".
- Tussen 50% en 68% (68 incl.): "Voldoende".
- Tussen 68% en 75% (75 incl.): "Onderscheiding".
- Tussen 75% en 85% (85 incl.): "Grote onderscheiding".
- Meer dan 85%: "Grootste onderscheiding".

Test je klasse door enkele objecten in je main en te onderzoeken of deze de juiste graden op het scherm printen. Bijvoorbeeld:

```
Rapport mijnpunten = new Rapport();
mijnpunten.Percentage = 65;
mijnpunten.PrintGraad();
```

9.5 OEFENINGEN — 1 / 7

Bibliotheek*

Boeken in een bibliotheek mogen maximum 14 dagen uitgeleend worden. Schrijf een console-applicatie om de volgende gegevens te tonen door middel van een klasse BibBoek:

- de naam van de ontlener, die werd ingelezen (autoproperty).
- de datum van vandaag (autoproperty met private set). Gebruik uiteraard de DateTime klasse.
- de datum dat het boek ten laatste terug moet ingeleverd worden (readonly property).

Nummers

Maak een klasse Nummers. Deze klasse bevat 2 getallen (type int) die via een autoproperty kunnen aangepast worden. Er zijn 4 methoden:

- Som: geeft de som van beide getallen terug.
- Verschil: geeft het verschil van beide getallen terug.
- Product: geeft het product van beide getallen terug.
- Quotient: geeft de deling van het eerste door het tweede getal terug; toon een foutboodschap op het scherm indien er een deling door nul zal gebeuren.

Kobo Desktop

SHOP KOBO MY BOOKS EREADER SETUP *Zie Scherp Scher...*

Toon in je main aan dat je code werkt (op de volgende pagina tonen we een voorbeeld).

Volgende code zou bijvoorbeeld onderstaande output moeten geven:

```
Nummers paar1 = new Nummers();
paar1.Getal1 = 12;
paar1.Getal2 = 34;
Console.WriteLine($"Paar: {paar1.Getal1}, {paar1.Getal2}");
Console.WriteLine($"Som = {paar1.Som()}");
Console.WriteLine($"Verschil = {paar1.Verschil()}");
Console.WriteLine($"Product = {paar1.Product()}");
Console.WriteLine($"Quotient = {paar1.Quotient()}");
```

Output:

```
Paar: 12, 34
Som = 46
Verschil = -22
Product = 408
Quotient = 0,352941176470588
```

BankManager*

Deel 1

We maken een Rekening klasse die kan gebruikt worden om de bankrekening van een klant voor te stellen. Deze heeft volgende zaken:

Kobo Desktop

SHOP KOBO MY BOOKS EREADER SETUP *Zie Scherp Scher...*

ZIE SCHERP SCHERPER - 2E EDITIE

- Een instantievariabele van het type int genaamd balans. Deze variabele houdt het totale bedrag bij dat op de rekening staat.
- 2 autoproperties van type string namelijk NaamKlant en RekeningNummer.
- 1 readonly property Balans die de balans teruggeeft.

Voorzie 3 methoden:

1. HaalGeldAf: bepaald bedrag (als parameter type int) wordt van de balans verwijderd.
2. StortGeld: bepaald bedrag (als parameter type int) wordt op de rekening gezet en aan balans toegevoegd.
3. ToonInfo: het totale bedrag op de rekening wordt getoond op het scherm, alsook de naam van de klant en het rekeningnummer (*ook de staat wanneer je deel 2 hebt gemaakt wordt getoond*).

Pas de HaalGeldAf methode aan zodat als returntype het bedrag (int) wordt teruggegeven. Indien het gevraagde bedrag meer dan de balans is dan geef je al het geld terug dat nog op de rekening staat en toon je in de console dat niet al het geld kon worden gegeven (error die verschijnt: Rekening leeg nu.)

Maak 2 instanties van het type Rekening aan en toon aan dat je geld van de ene Rekening aan de andere kunt geven, als volgt:

```
1 //rekening 2 geeft 300 euro aan rekening 1
2 rekening1.StortGeld(rekening2.HaalGeldAf(300));
```

9.5 OEFENINGEN — 4 / 7

Test je klasse.

1. Nieuwe klant aanmaken
2. Status van bestaande klant tonen
3. Geld op een bepaalde Rekening zetten
4. Geld van een bepaalde Rekening afhalen
5. Geld tussen 2 Rekeningen overschrijven

Voorzie extra functionaliteit naar keuze.

Deel 2

Voeg aan de Rekening-klasse een autoproperty,genaamd Staat, met private set toe van het type RekeningStaat toe, dat een enumeratie bevat. De Rekening kan in volgende staten zijn Geldig, Geblokkeerd. Een rekening is Geldig wanneer een nieuwe rekening wordt geopend.

Maak een bijhorende publieke methode waarmee je de Rekening van staat kunt veranderen. Deze methode (noem ze VeranderStaat) vereist geen parameters. Telkens je ze aanroept wordt de staat omgewisseld. Als dus het object momenteel op Geldig stond, dan wordt ze nu Geblokkeerd en omgekeerd.

Indien een persoon geld van of naar een Geblokkeerde rekening wil sturen dan zal er een error op het scherm verschijnen, namelijk Gaat niet. Rekening

geblokkeerd. Idem bij de StortGeld methode.

Indien de HaalGeldAf methode wordt aangeroepen en er werd meer geld afgehaald dan de balans dan zal de rekening ook op Geblokkeerd gezet worden na het verschijnen van de foutboodschap ("Rekening leeg nu").

Persoon

Ontwerp en implementeer een klasse Persoon met 2 autoproperties (string) Achternaam en Voornaam.

Voeg bovendien een full property GeboorteDatum toe (type DateTime). De geboortedatum kan enkel waarden tussen 1/1/1990 en vandaag (moment dat code wordt uitgevoerd) liggen. Indien dit niet de situatie is, wordt de huidige datum van uitvoeren gebruikt.

Definieer vervolgens een methode BerekenLeeftijd, die op basis van de ingestelde geboortedatum en de huidige datum (dat programma wordt uitgevoerd) de leeftijd van de persoon teruggeeft als int.

Verjaardag

Maak een applicatie die aan de gebruiker z'n geboortedatum vraagt (dag/maand, bv 18/3). Vervolgens toont het programma op welke dag hij volgend jaar jarig zal zijn (Monday, Tuesday, enz.) én hoeveel dagen hij nog

moet wachten. Uiteraard gebruik je DateTime voor deze applicatie.

Je hebt geen eigen klasse nodig. Doel van deze oefening is dat je leert werken met de bestaande DateTime en TimeSpan klassen. Je mag dus alles in de Main schrijven.

Volgende methode geeft de naam (als string) van de dag terug in de taal van het systeem waarop je applicatie draait:

```
System.Globalization.DateTimeFormatInfo.CurrentInfo.GetDayName();
```

Voorbeeld invoer en uitvoer:

```
Wanneer is je verjaardag (d/m, bv 18/3)
20/5
Je bent over 124 dagen jarig op een Dinsdag
```

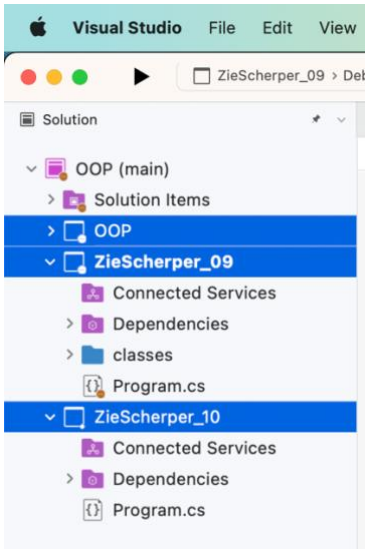
10 Opdracht 10 - Zie Scherp Scherper H10

We gaan verder met de online tutorial *Zie Scherp Scherper* en wel met *H10 Geheugenmanagement, uitzonderingen en namespaces*.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_10** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP en ZieScherper_09, en het nieuwe project **ZieScherper_10**:



Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_10**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

10.1 Stap 1 - Stack en Heap

Lees het eerste deel van dit hoofdstuk genaamd *Stack en Heap*:
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h10-geheugenmanagement-uitzonderingen-en-namespaces/6_memorymanagement

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Zorg dat de beschreven code van de method **VerhoogParameter()**, waarmee de werking van *value types* wordt gedemonstreerd, in je Program class terecht komt. Dus niet alleen de method zelf, maar ook de aanroep ervan. Maak van de method een *private static void* method, zodat je deze in je Main() kan aanroepen.
2. Maak een nieuwe class **Student** en neem daarin een *auto property* **Naam** op, type **string** en visibility **public**. In de constructor definieer je een *input parameter* **naam** van het type **string**, en zorg je dat de *auto property* Naam de waarde van de input parameter krijgt.
3. Maak zoals beschreven op de webpagina 2 objecten aan van de *Student* class, en zorg daarna m.b.v. de assignment operator (=) dat zowel de 1e instantie als de 2e instantie 'naar hetzelfde object in de heap verwijzen' (dit laatste wordt letterlijk genoemd op de webpagina).
4. Print de namen van de studenten naar de console, zowel vóór de bovengenoemde assignment als ná de assignment, waarmee de werking van *reference types* wordt gedemonstreerd.

   De verwachte output in je console ziet er als volgt uit:

   ```
   Naam student a: Abba
   Naam student b: Queen
   Naam student a: Abba
   Naam student b: Abba
   ```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 10.1**.

10.2 Stap 2 - Objecten en methoden

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht10_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Objecten en methoden:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h10-geheugenmanagement-uitzonderingen-en-namespaces/6b_objectenenmethoden

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de genoemde code van de class **Meting** op in je C# project (in een nieuwe class file!) met daarin de method **ToonMetingInKleur()**. Maak in de Main() method zoals beschreven 2 objecten (m1 en m2) aan van de class en roep op beide objecten de method ToonMetingInKleur() aan.
2. Voeg ook de code van de method **VoegMetingToeEnVerwijder()** toe aan de Meting class. Pas in de Main() method zoals beschreven de property waardes van de 2 objecten (m1 en m2) van de Meting class aan. Roep de nieuwe method aan op het eerste object (m1) en geef hierbij het tweede object als parameter mee.
   Neem het keyword `ref` op in de *signature* én de aanroep van de nieuwe method (**extra stap**, staat niet in de web tutorial).
3. Voeg de code van de method **GenereerRandomMeting()** toe aan de Meting class. Roep in de Main() method zoals beschreven de nieuwe method aan op het eerste object (m1) en stop het resultaat in een nieuw object (m3) van de Meting class. Print beide properties van het nieuwe object (extra stap, staat niet in de tutorial).
4. Neem de genoemde code van de class **Mens** op in je C# project (in een nieuwe class file!) met daarin de method **PlantVoort()**. Maak in de Main() method zoals beschreven 3 objecten aan van de class, 1 maal m.b.v. het `new` keyword en 2 maal door gebruik te maken van de PlantVoort() method.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand, waarbij de eerste 2 regels worden getoond in een zelf gekozen kleur:

```
26°C gemeten door: Lieven Scheire
34°C gemeten door: Ann Dooms
Temperatuur m1: 31 en m2: 0
Temperatuur m3: 52, opgemeten door Lieven Scheire Junior
```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 10.2**.

10.3 Stap 3 - Null en NullReferenceException

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht10_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Null en NullReferenceException:* https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h10-geheugenmanagement-uitzonderingen-en-namespaces/nullreference

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de genoemde code bij het kopje 'NullReferenceException' op in je Program class en check of je deze exception ook krijgt bij het runnen van de applicatie.
2. Los het exception probleem op de manier die in de tutorial wordt beschreven, bij het kopje 'NullReferenceException voorkomen'.
3. Voeg de code van de method **ZoekStudent()** toe aan de Student class en maak de method *public* zodat je deze in je *Main()* kan aanroepen. Maak in je Main() method een array van type Student met 2 of 3 objecten en roep de nieuwe method hiermee aan, de eerste keer met een geldige studenten naam (bestaand in de array), de twee keer met een ongeldige naam. Print het resultaat van beide zoek acties naar de console, dit kan per student in één statement! [2]:

   ```
   Naam: Jan
   ```

[2] Zie ook Google of de AI: *Ternary Operator*. Een goed uitleg kun je i.i.g. vinden op W3 Schools: https://www.w3schools.com/cs/cs_conditions_shorthand.php

```
Naam niet gevonden
```

4. Update de **PlantVoort()** method van de **Mens()** class (die je in de vorige opdracht hebt gemaakt) zoals beschreven bij het kopje *Bevalling in C# met ouders*, incl. de in de begeleidende tekst genoemde *auto property* **Geslacht** van het type **enum Geslachten**.
   Voeg de beschreven code toe aan je Main() method waarin een baby object van de Mens() class wordt 'gemaakt' op basis van een mama en een pappa Mens object.
   Print het resultaat naar de console:
   ```
   Baby gemaakt met Geslacht Man en MaxLengte 180
   ```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 10.3**.

10.4 Stap 4 - Namespaces en using

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht10_3()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Namespaces en using:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h10-geheugenmanagement-uitzonderingen-en-namespaces/namespaces

Voer nadat je alles hebt gelezen de volgende stap uit:

> Voer de tip uit die bij het kopje *Ontbrekende namespaces terugvinden* staat beschreven. Verander de namespace in de file waarin je de class Student() hebt geïmplementeerd tijdelijk in een andere naam (daar zal nu '.classes' staan als het goed is), bv:
> ```
> namespace ZieScherper_10.xyz // hier stond .classes
> ```
> Los de foutmelding op die hierdoor in je Program class ontstaat op de plek waar je de Student() class gebruikt, en wel door de IDE **automatisch** een using statement te laten genereren.

Als dit is gelukt kun je de tijdelijke aanpassingen weer ongedaan maken, een *commit* en *push* van deze test is niet nodig.

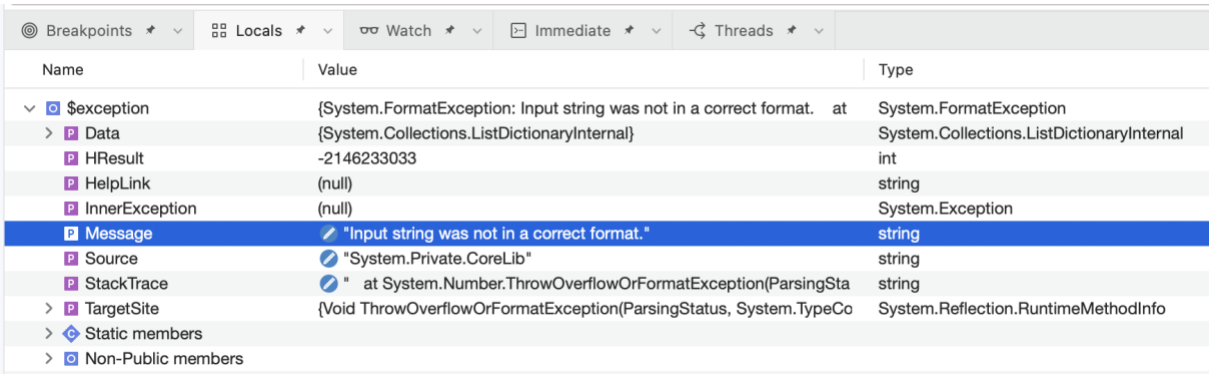
10.5 Stap 5 - Exception handling

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht10_4()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Exception handling:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h10-geheugenmanagement-uitzonderingen-en-namespaces/0_exceptionhandling

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de *try-catch* code die bij het kopje *Een try catch voorbeeld* staat beschreven op in je Program class. Zorg dat in het catch gedeelte een *FormatException* wordt afgehandeld.
   Run de code, zorg dat er een exception optreedt, en check of de exception netjes wordt afgehandeld.
2. Bij het kopje *Een exception genereren met throw* wordt een voorbeeld van een *throw* statement gegeven. Voeg een nieuw *try-catch* statement toe aan je code, plaats in het try gedeelte dit *throw* statement, en print in het catch gedeelte de *exception property* **Message** naar de console.
   Run de code en check of de exception netjes wordt afgehandeld.
3. Breidt, zoals beschreven bij het kopje *Meerdere catchblokken*, het eerste *try-catch* statement in je code uit met een tweede catch blok. Zorg dat deze, naast de aanwezige *FormatException,* ook een 'normale' (basisklasse) *Exception* afhandelt.
4. Zet een breakpoint in het eerste *try-catch* statement in je code, en wel in het catch gedeelte waar de *FormatException* wordt afgehandeld. Run de code en zorg dat er een exception optreedt. Analyseer de *exception properties*, waaronder de **Message** property, die in de debugger worden getoond:



Breakpoints | Locals | Watch | Immediate | Threads

Name	Value	Type
$exception	{System.FormatException: Input string was not in a correct format. at	System.FormatException
Data	{System.Collections.ListDictionaryInternal}	System.Collections.ListDictionaryInternal
HResult	-2146233033	int
HelpLink	(null)	string
InnerException	(null)	System.Exception
Message	"Input string was not in a correct format."	string
Source	"System.Private.CoreLib"	string
StackTrace	" at System.Number.ThrowOverflowOrFormatException(ParsingSta	string
TargetSite	{Void ThrowOverflowOrFormatException(ParsingStatus, System.TypeCo	System.Reflection.RuntimeMethodInfo
Static members		
Non-Public members		

Lees het volgend deel van dit hoofdstuk genaamd *Waar exceptions plaatsen?:*

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h10-geheugenmanagement-uitzonderingen-en-namespaces/waarplaatsen

Voer nadat je alles hebt gelezen de volgende stap uit:

5. Breidt zoals beschreven het eerste *try-catch* statement in je code uit met een *finally* blok en zet hierin een willekeurig print statement, dus zoiets als 'Finally blok wordt uitgevoerd'.
   Run de code, zorg dat er een exception optreedt, en check of het finally blok netjes wordt afgehandeld.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
Voer een getal in: xyz
Verkeerde invoer! Input string was not in a correct format.
Finally blok wordt uitgevoerd
Exception opgetreden: Wow, dit loopt fout
```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 10.5**.

10.6 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Oefening 'BankManager 2', deel a).

In de **Program** class een nieuwe method **SimuleerOverdracht()**, test method **Opdracht10_6a()**, commit **Opdracht 10.6a**.

Maak gebruik van de bestaande **Rekening()** class die je in H9 hebt gemaakt (d.m.v. een *project reference* [3]), dus NIET kopiëren. Verwachte output in je console zoiets als onder, die de rekening info voor én na de transacties laat zien:

```
Naam:           Pietje
Rekeningnummer: 1234
Staat:          Geldig
Balans:         €1000

Naam:           Klaasje
Rekeningnummer: 5678
Staat:          Geldig
Balans:         €1000

Transactie 1 van Pietje naar Klaasje voor bedrag van: €15
Transactie 2 van Klaasje naar Pietje voor bedrag van: €45
Transactie 3 van Pietje naar Klaasje voor bedrag van: €20
Transactie 4 van Klaasje naar Pietje voor bedrag van: €10
```

[3] Als je niet weet hoe je in MS Visual Studio een *project reference* maakt, zoek het even op via Google of de AI. Tip: het heeft te maken met de 'Dependencies' map in je project.
Verder: mogelijk staat de visibility van je Rekening class op de default 'internal'. Om deze class beschikbaar te maken in andere projecten moet je de class 'public' maken.

```
Transactie 5 van Pietje naar Klaasje voor bedrag van: €25

Naam:           Pietje
Rekeningnummer: 1234
Staat:          Geldig
Balans:         €995

Naam:           Klaasje
Rekeningnummer: 5678
Staat:          Geldig
Balans:         €1005
```

b) Oefening 'BankManager 2', deel b).

In de **Program** class een nieuwe method **CreeerTienerRekening()**, test method **Opdracht10_6b()**, commit **Opdracht 10.6b**. Maak wederom gebruik van de bestaande Rekening() class die je in H9 hebt gemaakt (d.m.v. een *project reference*), dus NIET kopiëren. Verwachte output in je console zoiets als onder, die de rekening info na het creëren van de studentenrekening laat zien:

```
Naam:           Jantje
Rekeningnummer: 1234
Staat:          Geldig
Balans:         €50
```

10.6 Oefeningen

Er zijn geen aparte Exception handling oefeningen. Het is een goede gewoonte dat je vanaf nu steeds goed nadenkt waar in je programma's je Exception handling kan toepassen.

Bankmanager 2

Breidt de bankmanager oefening uit het vorige hoofdstuk uit met volgende functionaliteiten:

- Voorzie in je programma een methode `SimuleerOverdracht`. Je kan aan deze methode 2 `Rekening` objecten meegeven. Vervolgens zal de methode 5x een willekeurig bedrag van de ene naar de andere rekening sturen, hierbij wisselen de rekeningen om de beurt wie verzender en wie ontvanger is. Wanneer de methode klaar is wordt er niets teruggestuurd.
- Maak een methode `CreeerTienerRekening` in je programma. Deze methode geeft een nieuwe rekening terug waar de balans reeds op

50 staat. De methode aanvaardt 1 parameter: de naam van de klant, dat vervolgens in het nieuwe object wordt ingesteld.

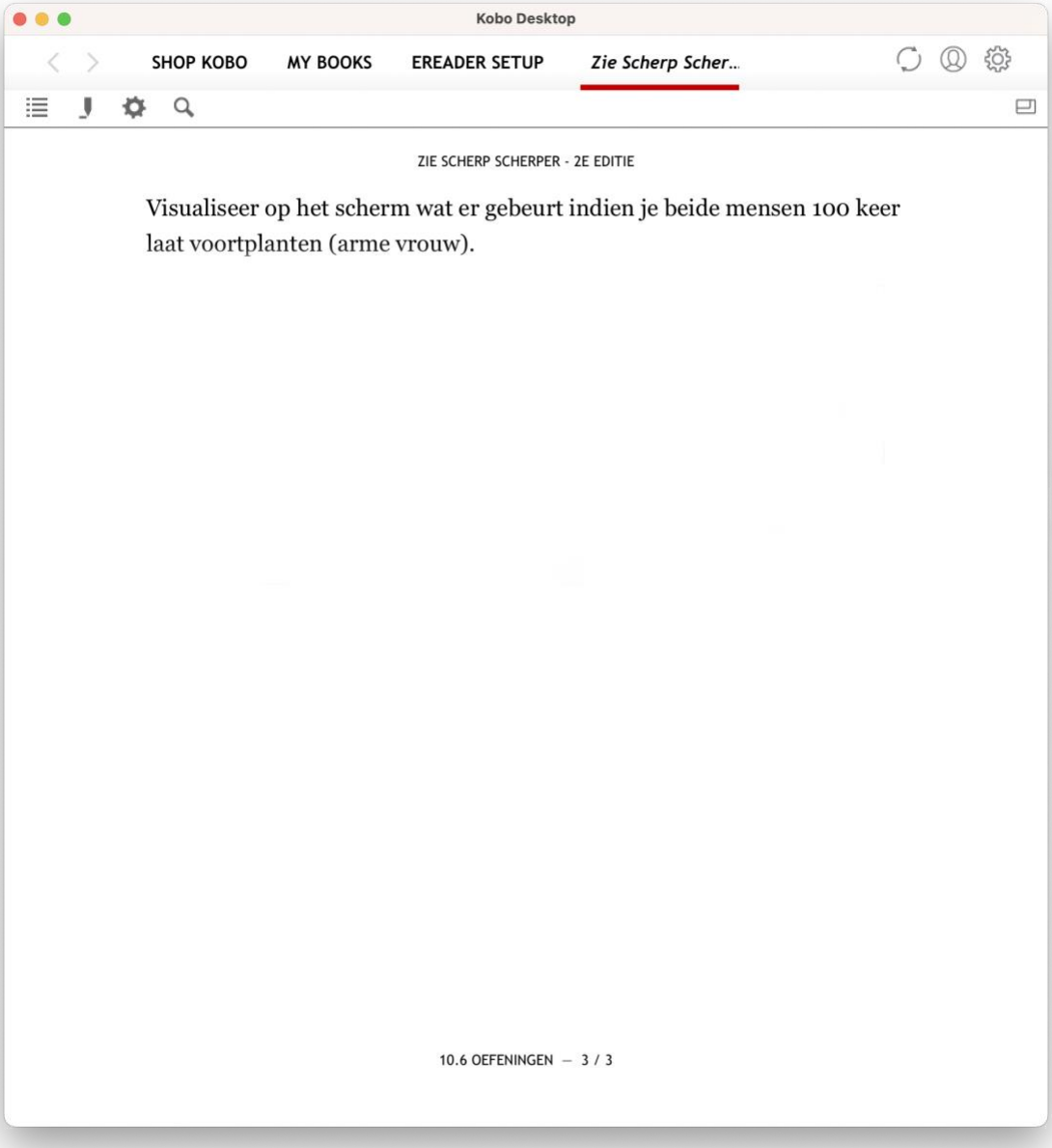
Voortplanten*

Laten we het voorbeeld van de klasse Mens en de methode PlantVoort uitbreiden. Een Mens wordt gedefinieerd door een geslacht (man of vrouw), kleur_ogen (als enum) en een maxlengte. Maak 2 Mens objecten aan en geef hen allemaal verschillende eigenschappen, maar zorg ervoor dat één een vrouw is en de ander een man. We passen nu de PlantVoort methode aan: de baby heeft als lengte de gemiddelde lengte van de man en vrouw. De oogkleur is 50% van de tijd die van de man en 50% van de tijd die van de vrouw. Het geslacht is willekeurig.

Introduceer echter ook een mutatie mogelijkheid: 10% van de tijd zal een baby een totaal willekeurige oogkleur krijgen, die niet noodzakelijk die van één van de ouders is.

Voorzie een methode ToonMens. Deze vat de informatie van een Mens-object in één zin samen, hierbij zal de achtergrond van de tekst de kleur van de ogen zijn. De tekst die verschijnt is: maxlengte [in meter], geslacht

Bijvoorbeeld:
1.78 m, man


Kobo Desktop
SHOP KOBO
MY BOOKS
EREADER SETUP
Zie Scherp Scher...
ZIE SCHERP SCHERPER - 2E EDITIE
Visualiseer op het scherm wat er gebeurt indien je beide mensen 100 keer laat voortplanten (arme vrouw).
10.6 OEFENINGEN — 3 / 3

11 Opdracht 11 - Zie Scherp Scherper H11

We gaan verder met de online tutorial *Zie Scherp Scherper* en wel met *H11 Gevorderde klasseconcepten*.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_11** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP, ZieScherper_09 en ZieScherper_10, en het nieuwe project **ZieScherper_11**:

- OOP (solution)
  - OOP (project)
  - ZieScherper_09 (project)
  - ZieScherper_10 (project)
  - **ZieScherper_11** (project)

Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_11**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

11.1 Stap 1 - Constructors en de default Constructors

Lees het eerste deel van dit hoofdstuk genaamd *Constructors en de default Constructors*: https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h11-gevorderde-klasseconcepten/1_constructors

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. In paragraaf 10.1 heb je al een **Student** class gemaakt en deze voorzien van een constructor, en wel eentje met een input parameter voor de naam van de student. Bekijk de code van deze constructor nogmaals (in het ZieScherper_10 project), en ook hoe je in de Program class instanties van deze class hebt gemaakt.
2. Voeg zoals beschreven in de tekst een tweede *default constructor* toe aan de Student class, ditmaal dus zónder parameters. Zorg dat je in de nieuwe constructor de bestaande *auto property* **Naam** een willekeurige *hard coded* value geeft.
3. Zorg ervoor dat je in de Program class m.b.v. een *using* statement de **Student** class uit het ZieScherper_10 project kan gebruiken.
   Als je niet meer weet hoe dat moet kijk dan even in je code van de 'BankManager' oefening uit het ZieScherper_10 project (paragraaf 10.6), waarin je hetzelfde hebt moeten doen om de 'Rekening' class (uit het ZieScherper_09 project) te kunnen gebruiken.
4. Nu je twee constructors hebt, test je deze door in de Main() twee instanties van de Student class te maken waarbij je gebruik maakt van beide constructors, en print daarna de naam van beide studenten naar de console, dus zoiets als:
   ```
   Naam student: Pietje
   Naam student: Jantje
   ```

Indien je test succesvol is verlopen, commit en push je de code naar je repo met als message: **Opdracht 11.1**.

11.2 Stap 2 - Overloaded Constructors

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht11_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Overloaded Constructors:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h11-gevorderde-klasseconcepten/2_overloadedconstructor

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. We gaan nog wat meer code toevoegen aan de Student class uit hoofdstuk 10 (zie kopje 'Meerdere overloaded constructors'). Maak, naast de bestaande property Naam, een tweede public property genaamd **IsWerkStudent** van het type **bool** (in het boek staat het genoemde type fout!).
2. Voeg zoals beschreven in de tekst een tweede *overloaded constructor* toe aan de Student class, ditmaal dus met twee parameters: **naam** van type string en **isWerkStudent** van type bool.
3. Test je nieuwe constructor door een twee instanties van de Student class te maken, met verschillende waardes voor de boolean parameter, en print daarna de naam én de waarde van de isWerkStudent property van beide studenten naar de console, dus zoiets als:

   ```
   Student Pietje is wel een werkstudent
   Student Jantje is geen werkstudent
   ```

4. Neem de genoemde code van de class **Microfoon** op in je C# project (in een nieuwe class file!), maar dan wel de herschreven versie die gebruik maakt van de **this()** aanroep. Voeg ook de daarna beschreven vierde constructor toe aan de nieuwe class.
5. Test de *constructor overloading* door twee instanties van de Microfoon class te maken, door (zoals ook staat beschreven) voor beide objecten de vierde versie van de constructor aan te roepen, en print daarna waardes van beide properties van de microfoon naar de console, dus zoiets als:

   ```
   Microfoon Wit Product is wel uitverkocht
   Microfoon Wit Product is niet uitverkocht
   ```

6. In het boek wordt in 4 stappen beschreven hoe de code van de *overloaded constructors* van de Microfoon klasse, als gevolg van bovenstaande instantiatie, wordt doorlopen. Loop nu zelf met de debugger stap voor stap door de code en controleer of je dezelfde waardes ziet in de debugger als in het boek:

```
Microfoon.cs    Program.cs
Microfoon › Microfoon(bool isUitverkochtIn)
 9          // Constructors hergebruiken met this()
10          public Microfoon(string merkIn,
11              bool isUitverkochtIn)
12          {
13              Merk = merkIn;
14              IsUitverkocht = isUitverkochtIn;
15          }
16          public Microfoon(string merkIn) ...
19          public Microfoon() ...
22
23          // aanroep: new Microfoon(true)
24          public Microfoon(bool isUitverkochtIn) :
25              this("Bovarc", isUitverkochtIn)
26          {
27              Merk = "Wit Product";
28          }
29
30          public string Merk { get; set; }
31          public bool IsUitverkocht { get; set; }
32      }
33  }
```

Breakpoints | Locals | Watch | Immediate | Threads | Search Results

Name	Value	Type
this	{ZieScherper_11.Microfoon}	ZieScherper_11.Microfoon
IsUitverkocht	true	bool
Merk	"Bovarc"	string
isUitverkochtIn	true	bool

7. Voer de beschreven aanpassing aan de **Balletje** class uit hoofdstuk 9 (paragraaf 9.1) uit (kopje 'Pong met constructors'). Oftewel, voeg de twee constructors toe aan de class en roep deze beide vervolgens aan op de plek waar je de objecten aanmaakt.

Indien al je testen succesvol zijn verlopen, commit en push je de code naar je repo met als message: **Opdracht 11.2**.

11.3 Stap 3 - Object Initializer Syntax

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht11_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Object Initializer Syntax:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h11-gevorderde-klasseconcepten/2_objectinitsyntax

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Maak twee nieuwe instanties van de Microfoon class d.m.v. *Object Initializer Syntax*, en print daarna waardes van beide properties van de microfoon naar de console, dus zoiets als:

   ```
   Microfoon Mijn Merk is wel uitverkocht
   Microfoon Mijn Merk is niet uitverkocht
   ```

2. In het boek wordt beschreven hoe, als gevolg van bovenstaande instantiatie, eerst de code van de constructors en daarna de code van de setters wordt doorlopen. Loop nu zelf met de debugger stap voor stap door de code en analyseer of je de volgorde waarin je code wordt uitgevoerd terugziet in de debugger.

Indien al je testen succesvol zijn verlopen, commit en push je de code naar je repo met als message: **Opdracht 11.3**.

11.4 Stap 4 - Static

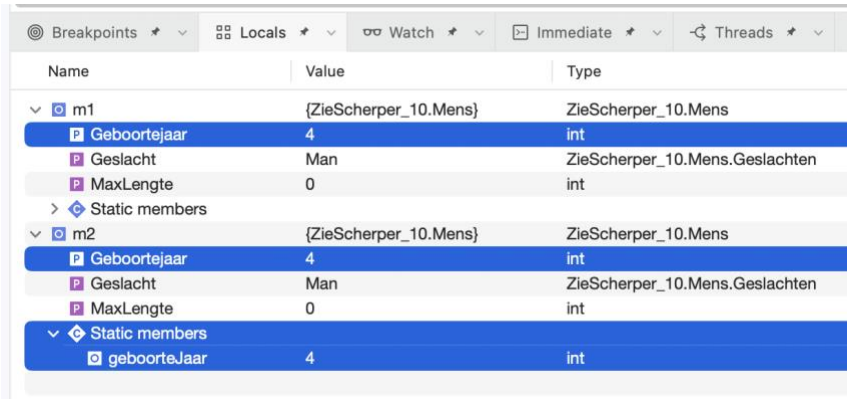
Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht11_3()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Static:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h11-gevorderde-klasseconcepten/5_static

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. In paragraaf 10.2 heb je al een **Mens** class gemaakt en deze in 10.3 verder uitgebreid. Bekijk de code van deze class nogmaals (in het ZieScherper_10 project), en ook hoe je in de Program class instanties van deze class hebt gemaakt.
2. Zorg ervoor dat je in de Program class m.b.v. een *using* statement de Mens class uit het ZieScherper_10 project kan gebruiken. Als het goed is heb je dit al gedaan in de eerste paragraaf uit dit hoofdstuk (11.1), waarin je dit hebt moeten doen voor de Student class (ook uit het ZieScherper_10 project).

3. Voeg aan de Mens class de beschreven method **Jarig()** en de bijbehorende properties voor **Geboortejaar** toe. Voeg ook de genoemde code toe die twee instanties van de Mens class maakt en de nieuwe method Jarig() aanroept, en wel aan je Main() method in de Program class. Run je project en test of je de (in het boek/webpagina) beschreven uitvoer krijgt (geboortejaar: 2 en 1).
4. Maak zoals beschreven van het private veld **geboorteJaar** een *static field*. Run je project nogmaals en test of je de beschreven output krijgt, die dus verschilt van vóór de aanpassing (geboortejaar: 4 en 4!). Loop ook met de debugger stap voor stap door de code en analyseer hoe de waarde van de public property **GeboorteJaar** veranderd in de debugger terwijl je voor verschillende Mens objecten (m1 en m2) de Jarig() method aanroept.



5. Neem de genoemde code van de *static class* **EpicLibrary** op in je C# project (in een nieuwe class file!) en roep in de Main() method zoals genoemd beide *static methods* aan.
6. Roep zoals beschreven de WriteLine() method van de *static class* **Debug** aan in je Main method, en kijk of je de tekst kan terug vinden in het Output Window.
7. Neem de genoemde code van de class **Fiets** op in je C#, voeg in de Main() method de beschreven test code toe, en run je project en test of je het verwachte resultaat krijgt (in het Output Window).

   Loop ook hier met de debugger stap voor stap door de code en analyseer hoe de waarde van het *static field* aantalFietsen veranderd in de debugger terwijl je door de code van de Fiets class loopt.

Indien al je testen succesvol zijn verlopen, commit en push je de code naar je repo met als message: **Opdracht 11.4**.

11.5 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Oefening 'Digitale kluis'.

Class **DigitaleKluis**, test method **Opdracht11_5a()**, commit **Opdracht 11.5a**.

Verwachte output in je console zoiets als onder:

```
De code van dit kluisje is -666
De code van dit kluisje is 1001

De code 1003 is geen geldige code (aantal pogingen = 1)
De code 1005 is geen geldige code (aantal pogingen = 2)
De code 1002 is geen geldige code (aantal pogingen = 3)
Deze code is geldig! Aantal pogingen = 4.
Gevonden! Code is 1001.
```

b) [Optioneel] Oefening 'Bibliotheek 2'.

Class **BibBoek**, test method **Opdracht11_5b()**, commit **Opdracht 11.5b**.

Maak gebruik van de bestaande **BibBoek()** class die je in H9 hebt gemaakt (d.m.v. een *project reference*), dus NIET kopiëren. Verwachte output in je console zoiets als:

```
De uitleendatum van het boek van Klaasje is 1/15 (mm/dd)
De uitleendatum van het boek van Pietje is 1/16 (mm/dd)
Kan niet uitlenen in de toekomst!
De inleverdatum van het boek van Klaasje is 1/30 (mm/dd)
De inleverdatum van het boek van Klaasje is veranderd
naar 1/23 (mm/dd)
```

11.4 Oefeningen

Digitale kluis

Basiskluis

Maak een klasse DigitaleKluis die we gaan gebruiken om een kluis voor te stellen.

De klasse heeft volgende elementen:

- Een fullproperty Code met private set. De get van deze property zal altijd -666 teruggeven, tenzij CanShowcode op true staat, in dit geval zal de effectieve code worden terug gegeven die in de bijhorende instantievariabele code staat.
- Een overloaded constructor die als parameter een geheel getal toelaat. Dit getal zal worden toegewezen aan Code en is dus de toegangscode
- Een autoproperty CanShowCode type bool, om aan te geven of de code van buitenuit kan gezien worden.
- Een read-only property CodeLevel van type int. Deze property zal het level van de code teruggeven (ongeacht CanShowCode). Het level is

eenvoudigweg de effectieve code gedeeld door 1000 als geheel getal (dus indien de code 500 is zal 0 worden teruggegeven, indien de code 2000 is wordt 2 teruggegeven, enz.)

- Een private variabele aantalPogingen om bij te houden hoe vaak de gebruiker geprobeerd heeft de code te vinden.
- Een methode TryCode die een geheel getal als parameter aanvaardt. De methode geeft een true terug indien de code correct was, anders false. Deze methode kan gebruikt worden om extern een code te testen , indien deze overeenkomt met de bewaarde code dan zal gemeld worden dat de code geldig is en wordt ook getoond hoeveel keer de gebruiker geprobeerd heeft Deze code is geldig. Aantalpogingen = 5. Indien de gebruiker -666 meegaf dan verschijnt er CHEATER op het scherm. Indien de gebruiker een foute code meegaf dan meldt de methode dat dit een foute code was ("geen geldige code") en wordt het aantal pogingen met 1 verhoogd. Deze methode kan maar 10 keer aangeroepen worden (ongeacht of de juiste code werd meegegeven of niet).Indien dus aantalPogingen 10 of meer is zal er op het scherm verschijnen Je hebt je 10 pogingen opgebruikt.Sorry..

Maak enkele Digitale Kluis objecten aan in je Main en test of je bovenstaande klasse correct is geïmplementeerd.

Kluizen kraken

Voeg aan de klasse een static methode BruteForce toe waar je een kluis-object aan kan geven. De methode zal een aantal random codes *bruteforcen* (met een loop die 10 verschillende getallen test) door telkens de TryCode methode van de meegegeven kluis aan te roepen. Wanneer toevallig de juiste code werd gevonden zal het aantal pogingen getoond worden.

Beeld je in dat Bruteforce een student is. Ik maak een kluis-object aan en vertel aan die student niet welke code ik heb gekozen. Vervolgens geef ik hem mijn object en zeg hem "Je krijgt 10 pogingen om met TryCode de kluiscode te raden. Succes!"

Bibliotheek deel 2

Neem de Bibliotheek oefening uit hoofdstuk 9 erbij. Zorg ervoor dat de BibBoek klasse volgende zaken heeft:

- Een default constructor die de Uitgeleend datum op gisteren zet.
- Een overloaded constructor die toelaat om de ontlener (string) en Uitgeleend (DateTime) als parameters mee tegen en in te stellen in het object. Indien de meegeven datum later is dan de huidige datum

dan zal er een uitzonderingen worden opgeworpen.

- Een static methode VeranderAlgemeneUitleenTermijn , die 1 int als parameter aanvaardt.De methode zorgt ervoor dat het aantal dagen dat je boek mag uitlenen verandert wordt naar de meegeven parameter. Tip: dit zal vereisen dat je een static instantievariabele hebt die je vervolgens in je InleverDatum getter gebruikt.

12 Opdracht 12 - Zie Scherp Scherper H12

We gaan verder met de online tutorial *Zie Scherp Scherper* en wel met *H12 Arrays en klassen*.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_12** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP, ZieScherper_09 tot en met ZieScherper_11, en het nieuwe project **ZieScherper_12**:

- OOP (solution)
  - OOP (project)
  - ZieScherper_09 (project)
  - ZieScherper_10 (project)
  - ZieScherper_11 (project)
  - **ZieScherper_12** (project)

Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_12**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

12.1 Stap 1 - Arrays van objecten

Lees het eerste deel van dit hoofdstuk genaamd *Arrays van objecten*:
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h12-arrays-en-klassen/7_arraysvanobj

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Zorg ervoor dat je in de Program class m.b.v. een *using* statement de **Student** class uit het ZieScherper_10 project kan gebruiken. Als je niet meer weet hoe dat moet kijk dan even in je code van het vorige hoofdstuk (paragraaf 11.1), daar heb je hetzelfde moeten doen.
2. Voeg aan de Student class een *auto property* **Geboortejaar** op, type **int** en visibility **public** en initialiseer de waarde op 2000.
3. Maak zoals beschreven in de Main() method een array van **Student** objecten aan en zorg ervoor dat de array ook daadwerkelijk objecten bevat (dus géén null waardes). Verander de naam van één van de studenten uit de array op één van de genoemde manieren.
4. Voeg zoals beschreven een tweede een array van **Student** objecten aan, ditmaal dus m.b.v. de *array initializer syntax*. Verander ook hier de naam van één van de studenten uit de array op één van de genoemde manieren.

Indien je project nog steeds goud bouwt, *commit* en *push* je de code met als message: **Opdracht 12.1**.

12.2 Stap 2 - List

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht12_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *List*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h12-arrays-en-klassen/4_list

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Definieer in de Main() method de eerste drie genoemde variabelen van de *generieke klasse* `List<>`. Vervang, in de derde definitie die een lijst van **Pokemon** objecten maakt, deze door een lijst genaamd **students** van de klasse **Student**. Je hebt dus nu 3 variabelen gedefinieerd: *alleGetallen*, *binaryList* en *students*.

2. Gebruik de **Add()** methode om aan elk van de bovengenoemde lijsten elementen toe te voegen.
   a) Voeg dus ook nieuwe studenten toe aan de desbetreffende lijst, en doe dat op beide manieren die worden beschreven, waaronder de *object initializer syntax* die je al eerder hebt toegepast.
   b) Voeg ook een 'werkstudent' toe, en maak hiervoor gebruik van de *overloaded constructor* uit de Student class (zie ook 11.2).
3. Print m.b.v. een `for` loop de waardes uit de gevulde lijsten naar de console. En wat betreft de student info: dit kan per student in één statement! [4]
4. Gebruik één of meer van de andere genoemde methods die je kunt gebruiken voor een variabele van de *generieke klasse* `List<>`. Loop met de debugger stap voor stap door de code en analyseer wat er met de inhoud van de lijsten gebeurt op het moment dat je deze methods aanroept.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand, waarin student 'Keesje' de toegevoegde 'werkstudent' is:

```
alleGetallen[0]: 1
alleGetallen[1]: 2
binaryList[0]: True
binaryList[1]: False
Student Klaasje is geen werkstudent en is geboren in het jaar 2000
Student Jantje is geen werkstudent en is geboren in het jaar 2000
Student Keesje is wel een werkstudent en is geboren in het jaar 2000
```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 12.2**.

[4] Zie ook Google of de AI: *Ternary Operator*. Een goed uitleg kun je i.i.g. vinden op W3 Schools: https://www.w3schools.com/cs/cs_conditions_shorthand.php

12.3 Stap 3 - Foreach en var

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht12_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Foreach en var:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h12-arrays-en-klassen/3_foreach

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Loop zoals beschreven m.b.v. een `foreach` loop door de waardes uit de studentenlijst (uit de vorige stap) en print de studentinfo naar de console. Je kunt dit dus zien als een verbeterde versie van de `for` loop die je in de vorige stap hebt gebruikt om door de studentenlijst te lopen. Zorg ervoor dat je dezelfde output krijgt.
2. Loop nogmaals door de studentenlijst en pas de inhoud van *het onderliggende object* aan door voor elke student het geboortejaar te verhogen met 1. Print daarna de studentinfo nogmaals naar de console.
3. Verander in de laatste `foreach` loop het type van de *iteration variabele* door `var`.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
Student Klaasje is geen werkstudent en geboren in het jaar 2000
Student Jantje is geen werkstudent en geboren in het jaar 2000
Student Klaasje is geen werkstudent en geboren in het jaar 2001
Student Jantje is geen werkstudent en geboren in het jaar 2001
```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 12.3**.

12.4 Stap 3 - Nuttige collection klassen

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht12_3()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Nuttige collection klassen:* https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h12-arrays-en-klassen/dict

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de genoemde code die de (first in, first out) *Queue<>* collectie demonstreert op in je Main() method en check of je de verwachte output krijgt. Voer daarna tweemaal de **Peek()** methode uit op de collection en print het resultaat. Print tenslotte de lengte van de collection, kijk even op Google als je niet (meer) weet welke method of property je hiervoor nodig hebt.
2. Neem de genoemde code die de (last in, first out) *Stack<>* collectie demonstreert op in je Main() method en check of je de verwachte output krijgt. Print daarna de lengte van de collection.
3. Neem de genoemde code die de (key-value pair) *Dictionary<>* collectie demonstreert op in je Main() method, dus tot en met de foreach die de inhoud van de klanten dictionary print, en check of je de verwachte output krijgt.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
Ik stond hier als eerste
Ik als tweede
Ik als laatste
Ik als laatste
```

```
De lengte van de queue is nu 1

Ik als laatste
Ik als tweede
De lengte van de stack is nu 1

123      : Tim Dams
6463     : James Bond
666      : The beast
700      : James Bond
```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 12.4**.

12.5 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Oefening 'Prijzen met foreach'.

   Geen class, test code in Main(); test method **Opdracht12_5a()**, commit **Opdracht 12.5a**.

   Het is voldoende om 5 prijzen i.p.v. 20 aan de gebruiker te vragen. Zorg wel voor een goede exception afhandeling voor het geval de gebruiker geen geldig getal opgeeft (zie ook 10.5). Verwachte output in je console zoiets als onder:

```
Geef prijs 1: 5.00
Geef prijs 2: 1.50
Geef prijs 3: xyz
Verkeerde invoer!
```

```
Geef prijs 3: 7.00
Geef prijs 4: 10.25
Geef prijs 5: 2.00

Prijzen groter of gelijk aan 5:
5
7
10.25
Gemiddelde is 5.15
```

b) Oefening 'Speelkaarten'.

Class **Speelkaart**, test method **Opdracht12_5b()**, commit **Opdracht 12.5b**. Maak van het genoemde field 'getal' (1 tot en met 13) ook een enumtype (aas, een, twee, etc), zodat bv 13 als 'Heer' wordt weergegeven en niet als '13'. Verwachte output in je console zoiets als onder (print alle kaarten):

```
> Getrokken kaart: Klaveren Drie (kaart 29 van 52 overgebleven)
Druk op een toets voor de volgende kaart
> Getrokken kaart: Harten Zes (kaart 19 van 51 overgebleven)
Druk op een toets voor de volgende kaart
...
> Getrokken kaart: Harten Vier (kaart 2 van 2 overgebleven)
Druk op een toets voor de volgende kaart
> Getrokken kaart: Schoppen Tien (kaart 1 van 1 overgebleven)
```

c) Oefening 'Computer-winkel*'.

Class **ComputerOnderdeel**, test method **Opdracht12_5c()**, commit **Opdracht 12.5c**. Verder in de **Program** class de nieuwe methods **ToonInfo_Alle()**, **ToonInfo_PrijsGroterDan400()**, **ToonInfo_OnderdelenInDoos()**, **ToonInfo_EvenIDPrijsGroterDan200()**, **Verwijder_InDoos()**, en **Verwijder_PrijsOnder100()** die je vanuit de main() aanroept.

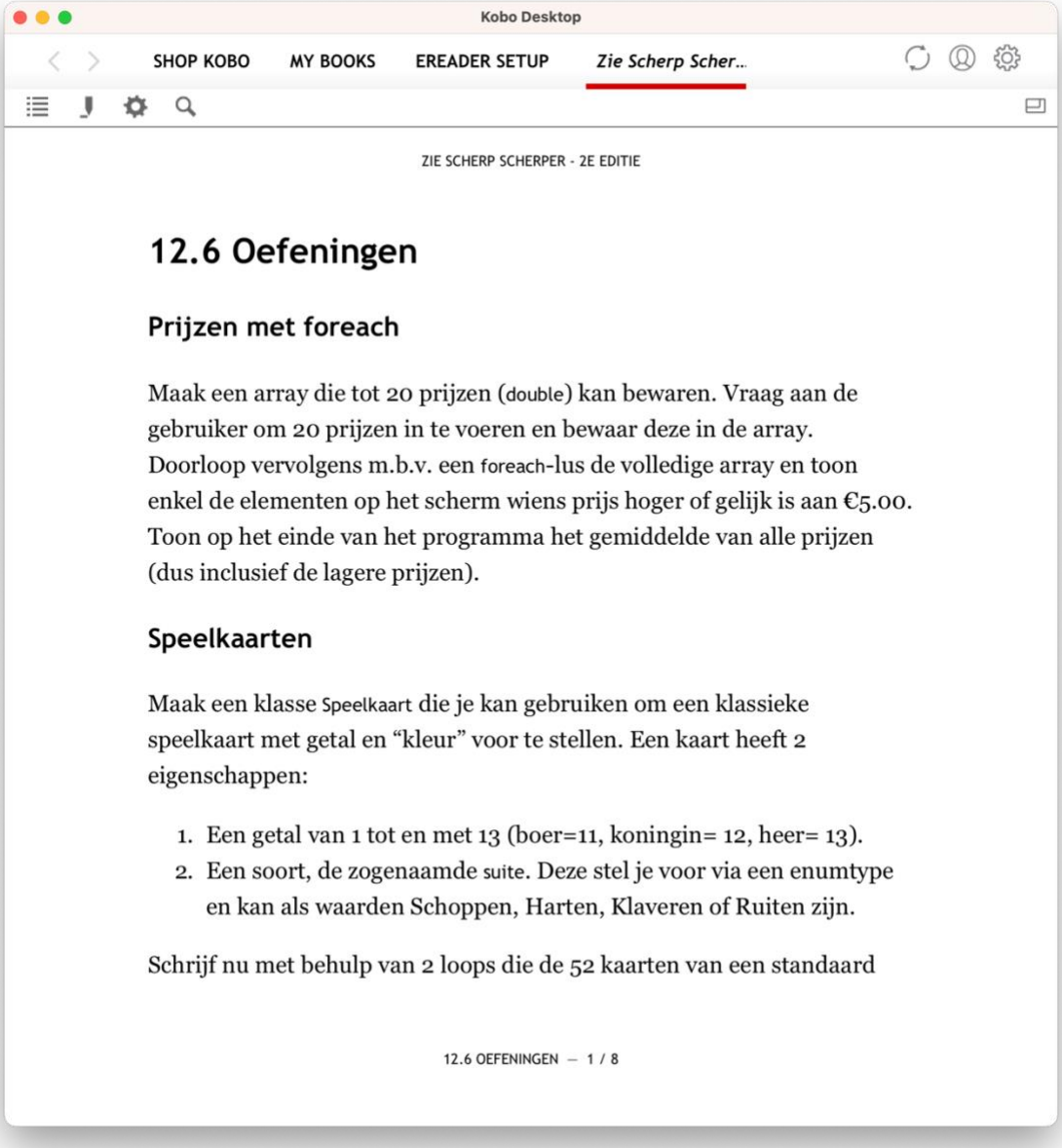
De deelopdracht bij het laatste kopje 'Finale' hoef je niet uit te voeren.

Verwachte output in je console zoiets als onder:

```
Alle onderdelen in de lijst:
```

```
Prijs: 742, ID: 461, InDoos: False
Prijs: 786, ID: 266, InDoos: False
Prijs: 510, ID: 645, InDoos: False
...

Alle onderdelen met een prijs > 400:
Prijs: 742, ID: 461, InDoos: False
Prijs: 786, ID: 266, InDoos: False
Prijs: 510, ID: 645, InDoos: False
...

Alle onderdelen in een doos:
Prijs: 57, ID: 110, InDoos: True
Prijs: 232, ID: 152, InDoos: True
Prijs: 740, ID: 755, InDoos: True
...

Alle onderdelen met een even ID & prijs < 200:
Prijs: 57, ID: 110, InDoos: True
Prijs: 199, ID: 522, InDoos: False
Prijs: 157, ID: 700, InDoos: False
...

Alle onderdelen die in een doos zitten verwijderen? (y/n) y
Prijs: 605, ID: 217, InDoos: True => Verwijderd
Prijs: 457, ID: 362, InDoos: True => Verwijderd
Prijs: 661, ID: 879, InDoos: True => Verwijderd
...

Onderdelen met een prijs lager dan 100 verwijderen? (y/n) y
Prijs: 3, ID: 937, InDoos: False => Verwijderd
Prijs: 4, ID: 621, InDoos: False => Verwijderd
Prijs: 49, ID: 530, InDoos: False => Verwijderd

Overgebleven onderdelen:
```

```
Prijs: 742, ID: 461, InDoos: False
Prijs: 786, ID: 266, InDoos: False
Prijs: 510, ID: 645, InDoos: False
...
```



Kobo Desktop

SHOP KOBO MY BOOKS EREADER SETUP *Zie Scherp Scher...*

ZIE SCHERP SCHERPER - 2E EDITIE

12.6 Oefeningen

Prijzen met foreach

Maak een array die tot 20 prijzen (double) kan bewaren. Vraag aan de gebruiker om 20 prijzen in te voeren en bewaar deze in de array. Doorloop vervolgens m.b.v. een foreach-lus de volledige array en toon enkel de elementen op het scherm wiens prijs hoger of gelijk is aan €5.00. Toon op het einde van het programma het gemiddelde van alle prijzen (dus inclusief de lagere prijzen).

Speelkaarten

Maak een klasse Speelkaart die je kan gebruiken om een klassieke speelkaart met getal en "kleur" voor te stellen. Een kaart heeft 2 eigenschappen:

1. Een getal van 1 tot en met 13 (boer=11, koningin= 12, heer= 13).
2. Een soort, de zogenaamde suite. Deze stel je voor via een enumtype en kan als waarden Schoppen, Harten, Klaveren of Ruiten zijn.

Schrijf nu met behulp van 2 loops die de 52 kaarten van een standaard

12.6 OEFENINGEN — 1 / 8

pak in een List<SpeelKaart> lijst plaatst.

Maak een applicatie die telkens een willekeurige kaart uit de *stapel* trekt en deze aan de gebruiker toont. De kaart wordt na het tonen ook uit de lijst verwijderd. Door met een willekeurig getal te werken hoef je dus je deck niet te schudden.

Computer-winkel*

Een firma heeft een grote lijst met computeronderdelen en wil hier de nodige filters op kunnen toepassen.

ComputerOnderdeel

De klasse ComputerOnderdeel bestaat uit volgende autoproperties:

- Prijs (int)
- ID (int)
- InDoos (bool)

Voorts heeft de klasse een default constructor die voorgaande autoproperties op willekeurige waarden instelt (prijs positief tot max 1000, ID een getal tussen 100 en 999)

De klasse heeft 1 methode ToonInfo die de 3 waarden van het object naar

het scherm stuurt:

```
Prijs: 845, ID: 45, InDoos: true
```

Filteren

Maak in je hoofdprogramma een List<ComputerOnderdeel> dat je vult met 100 willekeurige aangemaakte objecten.
Vervolgens:

- Roep de ToonInfo methode aan van ieder computeronderdeel in de lijst aan.
- Roep de ToonInfo methode aan van ieder computeronderdeel met een prijs boven de 400.
- Idem nu voor alle onderdelen die in een doos zitten.
- Idem nu voor alle onderdelen die een even ID hebben én wiens prijs onder de 200 ligt.

Tussen iedere filter toon je op het scherm wat de volgende lijst juist voorstelt (bv "Nu tonen we alle onderdelen in een doos").

Managen

Vraag nu aan de gebruiker wat er met de lijst moet gebeuren:

1. Alle objecten in een doos verwijderen
2. Alle objecten met een prijs kleiner dan 100 verwijderen.

Toon het resultaat van de aangepaste lijst (door de ToonInfo van ieder object in de lijst aan te roepen.)

Finale

Kan je hier een volledige applicatie van maken die een computerfirma als een soort inventaristool kan gebruiken (en dus met de nodige menu's en mogelijkheden om bijvoorbeeld een nieuw onderdeel toe te voegen)? Kijk zeker eens naar volgende oefening daaromtrent.

Bookmark Manager*

Maak een "bookmark manager". Deze tool zal in de console aan de gebruiker 5 favoriete sites vragen: naam en url. Vervolgens zal de tool alle sites in een lijst tonen met een nummer ervoor. De gebruiker kan dan de nummer intypen en de tool zal automatisch de site in een vereenvoudigde versie op het scherm tonen.

Je opdracht:

1. Maak een array of List waarin je 5 bookmark objecten kan plaatsen.

2. Vul de applicatie aan zodat de gebruiker een bestaand bookmark kan verwijderen en een bestaand bookmark kan aanpassen.
3. De gebruiker kan een bookmark selecteren (door de index ervan in te geven) en vervolgens zal deze site getoond worden op het scherm.

Hierna volgen enkele zaken die je zal nodig hebben.

```
class BookMark
{
  public BookMark(string naamIn, urlIn)
  {
     Naam = naamIn;
     URL = urlIn;
  }

  public string Naam { get; set; }
  public string URL { get; set; }
  public string DownloadSite()
  {
     var wc = new WebClient();
     string result = wc.DownloadString(URL);
     return GetPlainTextFromHtml(result);
  }

  private string GetPlainTextFromHtml(string html)
  {
     string htmlTagPattern = "<.*?>";
     string reg = "(\\<script(.+?)\\</script\\>)|(\\<style(.+?)\\</style\\>)";
     var regCss =
        new Regex(reg, RegexOptions.Singleline|RegexOptions.IgnoreCase);
     html = regCss.Replace(html, string.Empty);
     html = Regex.Replace(html, htmlTagPattern, string.Empty);
     html = Regex.Replace(html, @"^\s+$[\r\n]*", "", RegexOptions.Multiline);
     html = html.Replace(" ", " ");
```

Kobo Desktop

SHOP KOBO MY BOOKS EREADER SETUP *Zie Scherp Scher...*

ZIE SCHERP SCHERPER - 2E EDITIE

```
28        return html;
29    }
30 }
```

De GetPlanTextFromHtml() methode komt van volgende post:: **www.mercator.eu/mercator/std/info_aruba/reporting-hoe-gegevens-afdrukken-met-html-tags.html**.

Voorbeeld van hoe de Bookmark klasse zal werken:

```
1 BookMark u = new BookMark("Google", "https://www.google.be" );
2 Console.WriteLine(u.DownloadSite());
```

Kan je je eigen "console browser" maken?

Voortplanten en generaties

We gaan de voortplantings-simulator uit hoofdstuk 10 uitbreiden. Voorzie volgende constructors in de klasse Mens:

- Een default constructor die bovenstaande eigenschappen op willekeurige waarden instelt. Bij maxlengte is deze waarde 90% van de tijd tussen 150 en 210. 5% van de tijd is deze een getal tussen 40 en 150. 5% van de tijd een getal tussen 210 en 240.
- Een overloaded constructor waarbij de 3 eigenschappen als

12.6 OEFENINGEN — 6 / 8

parameter kunnen doorgegeven worden.

Maak 12 willekeurige mensen en plaats deze in een List. De eerste 6 mensen zijn vrouwen, de laatste 6 mensen zijn mannen. Alle mensen hebben willekeurige eigenschappen, enkel het geslacht is dus al bepaald. Laat nu de mensen voortplanten.

- De eerste helft mensen in de List zoeken een willekeurige partner van het andere geslacht in de andere helft lijst. En maakt hier een nieuwe mens mee. De man wordt meegegeven als parameter aan de Plantvoort methode van de vrouw.
- Plaats deze nieuwe mens in de lijst achteraan.

Generaties

Voeg een property Generatie toe aan je `Mens` die toelaat om te onthouden wanneer Mens is aangemaakt. Je kan deze via de constructor instellen. De eerste 12 mensen zijn generatie 1. Een nieuw mens krijgt als generatie de generatie van z'n ouder + 1 . Indien beide ouders van een verschillende generatie waren, dan krijgt het kind de generatie+1 van z'n ouder met de hoogste generatie van beide.

Herhaal de voortplantingscyclus voor 10 generaties (merk op dat ouders dus vermoedelijk kinderen zullen maken met hun eigen

kinderen/kleinkinderen. Niet over nadenken) en bekijk wat de resultaten zijn. Plaats deze simulatie in een static methode Simuleer die een List van mensen als parameter aanvaardt, alsook een getal dat aangeeft hoeveel generaties moet getest worden.

Houdt bij iedere Mens die je aanmaakt om welke generatie het gaat.

Toon aan de gebruiker hoeveel de gemiddelde lengte is, hoeveel percent welke kleur ogen heeft, etc.

Maak een applicatie waarbij de gebruiker experimenten kan doen om te zien wat er gebeurt na x generaties. Bijvoorbeeld:

- Wat als mannen erg klein zijn, vrouwen erg lang.
- Wat als mensen met blauwe ogen steeds die kleur aan hun kinderen doorgeven, na hoeveel generaties heeft iedereen blauwe ogen.

13 Opdracht 13 - Zie Scherp Scherper H13

We gaan verder met de online tutorial *Zie Scherp Scherper* en wel met *H13 Overerving*.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_13** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP, ZieScherper_09 tot en met ZieScherper_12, en het nieuwe project **ZieScherper_13**:

- OOP (solution)
  - OOP (project)
  - ZieScherper_09 (project)
  - ZieScherper_10 (project)
  - ...
  - **ZieScherper_13** (project)

Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_13**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

13.1 Stap 1 - Overerving intro

Lees het eerste deel van dit hoofdstuk genaamd *Overerving intro*:
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h13-overerving/0_overerving_intro

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Neem de genoemde code van de classes **Paard** en **Dier** (kopje 'Overerving in C#') op in je C# project in een nieuwe class file genaamd **Dier.cs**, en neem de instantiaties van deze classes en bijbehorende method aanroepen op in de Main() method van de Program class.
   Bouw en run je project en test of je de beschreven output krijgt. Zet het statement dat voor de foutmelding zorgt in commentaar.
2. Implementeer de beschreven klassehiërarchie in het kopje 'Overerving en het geheugen' in een nieuwe class file genaamd **Gebouw**.cs. De code hiervan staat dus niet in de tutorial, maar dat zou m.b.v. van je code uit de vorige stap moeten lukken. Je hoeft geen methods te definiëren, alleen de (lege) classes.
   Maak ook zoals beschreven objecten aan van de classes **Huis** en **Villa**, en plaats deze code in de Main() method van de Program class.
3. Definieer in de Dier class de beschreven *private field* **geboortejaar** (kopje 'protected') en in de Paard class de MaakOuder() method. Run je project, en los de foutmelding in de method op door van de field een *protected field* te maken.
4. Probeer in de Main() method de field **geboortejaar** van de Dier en Paard objecten een waarde te geven. Los de foutmelding op door (tijdelijk) de *accessibility level* van de field te veranderen (staat niet in de tutorial, dus Google indien nodig!).
   Maak die laatste (tijdelijke) verandering weer ongedaan, zodat de *accessibility* van geboortejaar weer de gewenste waarde *protected* krijgt.

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 13.1**.

13.2 Stap 2 - Constructors bij overerving

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht13_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Constructors bij overerving*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h13-overerving/3_constructors_inheritance

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de beschreven code van de classes **Soldaat** en **Veldarts** op in je C# project in een nieuwe class file genaamd **Soldaat.cs**, en neem de instantiatie van de Veldarts class op in de Main() method. Run je project en test of je de beschreven output krijgt. Loop als extra stap ook met de debugger door de code vanaf de instantiatie in de Main() method en analyseer de volgorde van de aanroep van beide constructors.
2. Voeg zoals beschreven aan de constructor van de Soldaat class een parameter **kanSchieten** van type **bool** toe. Je hebt nu van de *default constructor* een *overloaded constructor* gemaakt.
   Zorg nu met behulp van `base()` dat de *default constructor* van VeldArts een (fixed) boolean waarde kan doorgeven aan de *overloaded constructor* van de Soldaat class.
3. Voeg zoals beschreven een *overloaded constructor* aan de VeldArts class toe met een parameter **kanSchieten** van type **bool**, en dus incl. de `base()` die een (variabele) boolean waarde kan doorgeven aan de *overloaded constructor* van de Soldaat class.
4. Test de nieuwe *overloaded constructor* van de VeldArts class door in de Main() method een tweede instantiatie van de Veldarts class te maken, waaraan je ditmaal dus een (fixed) boolean waarde meegeeft.
   Loop ook hier als extra stap met de debugger door de code vanaf de tweede instantiatie en analyseer hoe de waarde van de boolean parameter tussen de constructors wordt doorgegeven.
   De verwachte output in je console zou zoiets moeten zijn als onderstaand:

   ```
   Soldaat is aangemaakt (kanSchieten: True)
   VeldArts is aangemaakt
   Soldaat is aangemaakt (kanSchieten: False)
   VeldArts is aangemaakt (kanSchieten: False)
   ```

5. Voeg aan je **Gebouw** class uit de vorige paragraaf de beschreven code betreffende de verdiepingen property toe, en aan de **Huis** class de code betreffende de verdiepingen

én tuin properties. Let op dat dit ook invloed heeft op de nu nog afwezige (!) constructors van de child classes **Appartement** en **Villa**, deze zul je overeenkomstig moeten aanpassen (Dit kan automatisch met behulp van de IDE! En wel m.b.v. 'Show potential fixes', en dan de optie ' Generate constructor').
Pas daarna in je Main() method de bestaande instantiaties van je Huis en Villa classes aan, want er worden nu parameters verwacht.
Controleer tenslotte of je project nog steeds goed bouwt.

6. Loop voor de laatste keer als extra stap met de debugger door de code vanaf de eerste aangepaste instantiatie en analyseer hoe de waarde van de parameters tussen de constructors wordt doorgegeven. Wat je in de debugger ziet zou overeen moeten komen met het plaatje uit de tutorial.

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 13.2**.

13.3 Stap 3 - Virtual en override

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht13_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Virtual en override:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h13-overerving/1_virtual_override

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de beschreven code van de classes **Vliegtuig** en **Raket** op in je C# project in een nieuwe class file genaamd **Vliegtuig.cs**, en neem de instantiaties van deze classes en bijbehorende method aanroepen op in de Main() method. Run je project en test of je de beschreven output krijgt.
2. Voeg aan de Raket class the *override* method **Vlieg()** toe. Run je project nogmaals en test of je de beschreven (en veranderde) output krijgt.

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 13.3**.

13.4 Stap 4 - Base keyword

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht13_3()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Base keyword:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h13-overerving/2_base

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de beschreven code van de classes **Restaurant** en **Frituur** op in je C# project in een nieuwe class file genaamd **Restaurant.cs**. Maak instanties van deze classes in de Main() method en roep op beide objecten de **PoetsAlles()** method aan (extra code, staat niet in de tutorial).
2. Loop met de debugger door de code vanaf de eerste instantiatie in de Main() method en analyseer de volgorde van de aanroep van de *virtual* en *overriden* methods, en hoe het **kosten** field van waarde veranderd.

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 13.4**.

13.5 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Oefening 'Ziekenhuis Deel 1'.

   Class **Patient**, test method **Opdracht13_5ab()**, commit **Opdracht 13.5a**. Maak om je class te testen een lijst met patiënten en gebruik hiervoor het **List<object>** type. (zie ook 12.2).

b) Oefening 'Ziekenhuis Deel 2'.

   Class **VerzekerdePatient** die je in dezelfde class file als Patient mag opnemen, dezelfde test method **Opdracht13_5ab()**, commit **Opdracht 13.5b**. Voeg, om je class te testen, objecten van de nieuwe class toe aan je bestaande lijst met patiënten.

Verwachte output na beide oefeningen in je console zoiets als onder:

```
Naam: Pietje, Kosten: 350 (Ziekenhuisuren: 15)
Naam: Jantje, Kosten: 150 (Ziekenhuisuren: 5)
Naam: Klaasje, Kosten: 250 (Ziekenhuisuren: 10)
Naam: Joske (verzekerd), Kosten: 315 (Ziekenhuisuren: 15)
Naam: Franske (verzekerd), Kosten: 135 (Ziekenhuisuren: 5)
```

13.6 Oefeningen

Ziekenhuis

Deel 1

Maak een basisklasse Patient die een programma kan gebruiken om de doktersrekening te berekenen.

Een patiënt heeft:

- Een naam.
- Het aantal uur dat hij in het ziekenhuis heeft gelegen.

Een virtual methode BerekenKost zal de totaalkost berekenen. Deze bestaat uit 50 euro + 20 euro per uur dat de patiënt in het ziekenhuis lag.

Maak een methode InfoFiche die steeds de naam van de patiënt teruggeeft gevolgd door het aantal uur en z'n kosten.

Deel 2

Maak een specialisatieklasse VerzekerdePatient. Deze klasse heeft alles dat

een gewone Patient heeft, echter de berekening van de kosten zal steeds gevolgd worden door een 10% reductie.

Toon de werking aan van deze klasse.

HiddenBookmark*

Voeg een HiddenBookmark klasse toe aan je bestaande Bookmark Manager applicatie van vorige hoofdstuk.

De HiddenBookmark is een Bookmark klasse die de DownloadSite methode override door VOOR en NA de gedownloade site string die je teruggeeft er de tekst **********INCOGNITO MODE********** aan toe te voegen.

Test wat er gebeurt als je al je bookmarks vervangt door HiddenBookmarks.

Pong Advanced

Kan je het Pong spel uit dit boek uitwerken, waarbij er echter verschillende soorten Balletje klassen zijn die allemaal van Balletje overerven? Maak de methoden Update en TekenOpScherm virtueel zodat je allerlei verschillende balletjes kunt verzinnen (grotere, andere kleur, ander soort update (denk maar aan teleportatie, versnellen en vertragen, enz.))

14 Opdracht 14 - Zie Scherp Scherper H14

We gaan verder met de online tutorial *Zie Scherp Scherper* en wel met *H14 Gevorderde overervingsconcepten*.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_14** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP, ZieScherper_09 tot en met ZieScherper_13, en het nieuwe project **ZieScherper_14**:

- OOP (solution)
  - OOP (project)
  - ZieScherper_09 (project)
  - ZieScherper_10 (project)
  - ...
  - **ZieScherper_14** (project)

Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_14**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

14.1 Stap 1 - System.Object

Lees het eerste deel van dit hoofdstuk genaamd *System.Object*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h14-gevorderde-overervingsconcepten/4_system_object

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Zorg ervoor dat je in de Program class m.b.v. een *using* statement de **Student** class uit het ZieScherper_10 project kan gebruiken. Als je niet meer weet hoe dat moet kijk dan even in je code van Opdracht 12 (paragraaf 12.1), daar heb je hetzelfde moeten doen.
2. Maak in je Main() method een instantie van de Student class, roep de **GetType()** method hierop aan, en zorg dat het resultaat van de **GetType()** method naar de console wordt geprint. Bouw en run je project en test of je de beschreven output krijgt.
3. Roep de **ToString()** method op het Student object aan, zorg dat het resultaat naar de console wordt geprint en controleer of je de beschreven output krijgt. Implementeer zoals beschreven in de Student class een *override* versie van de ToString() method. Run je project opnieuw en test of je de beschreven (en veranderde) output krijgt.
4. Implementeer zoals beschreven in de Student class een *override* versie van de **Equals()** method. Maak in je Main() method een tweede instantie van de Student class en vergelijk deze met het eerste Student object. Run je project test of je de verwachte output krijgt.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
ZieScherper_10.Student
Pietje (2000)
Object met inhoud 'Pietje (2000)' is niet gelijk aan 'Pietje (2001)'
```

Indien dat het geval is *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 14.1**.

14.2 Stap 2 - Abstract

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht14_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Abstract*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h14-gevorderde-overervingsconcepten/5_abstract

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de genoemde code van de *child classes* **Paard** en **Wolf**, en de *abstracte parent class* **Dier** op in je C# project in een nieuwe class file genaamd **Dier.cs**, en neem de instantiaties van deze classes en bijbehorende method aanroepen op in de Main() method van de Program class.
   Bouw en run je project en test of je de beschreven output krijgt. Zet het statement dat voor de foutmelding zorgt in commentaar.
2. Neem de beschreven definitie van de *abstract method* **MaakGeluid()** op in de parent class Dier, en los de foutmelding op die je nu krijgt op door (automatisch m.b.v. de IDE!) override methods voor dezelfde method toe te voegen aan de child classes. Roep de method aan op de instantiaties van deze classes in je Main() method.
3. Neem de beschreven definitie van de *abstract property* **MaxLeeftijd** op in de parent class Dier, en los de foutmelding op die je nu krijgt op door (automatisch m.b.v. de IDE!) override properties voor dezelfde method toe te voegen aan de child classes. Roep de property aan op de instantiaties van deze classes in je Main() method.

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 14.2**.

14.3 Stap 3 - Eigen exceptions maken

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht14_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Eigen exceptions maken:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h14-gevorderde-overervingsconcepten/1_eigenuitzondering

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de genoemde code van de method **ResultaatBerekening**() op in je Program class, en de aanroep ervan in de Main() method. Bouw en run je project en test of je de beschreven output krijgt.
2. Neem de genoemde code van de class **Timception** op in je C# project (in een nieuwe class file!), en zorg dat in je Main() method deze exception optreedt én wordt opgevangen. Je mag natuurlijk de naam van de class veranderen naar je eigen naam en ook de **extrainfo** string naar eigen behoeven aanpassen.

De verwachte output in je console zou na beide stappen zoiets moeten zijn als onderstaand. De tekst in het rood bestaat uit je eigen gedefinieerde exception string:

```
BOEM. ZWART GAT!
MyException extra info. ZieScherper_14.MyException: Exception of
type 'ZieScherper_14.MyException' was thrown.
...
```

Indien dat het geval is *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 14.3**.

14.4 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Oefening 'Boek'.

   Classes **Boek**, **TextBoek** en **KoffietafelBoek** in class file Boek.cs, test method **Opdracht14_4a()**, commit **Opdracht 14.4a**.

Verwachte output in je console zoiets als onder:

```
Een Boek - Boek Auteur (1234) _ 30
Een TextBoek - TextBoek Auteur (3456) _ 30
Een KoffietafelBoek - KoffietafelBoek Auteur (5678) _ 0
Omnibus van 'TextBoek Auteur, KoffietafelBoek Auteur' -
<auteur> (34565678) _ 15
```

b) Oefening 'Geometric figures'.

Classes **GeometricFigure**, **Rechthoek**, **Vierkant** en **Driehoek** in class file GeometricFigure.cs, test method **Opdracht14_4b()**, commit **Opdracht 14.4b**.

Maak als extra stap ook een **Cirkel** class, met een auto property **Straal**, waarmee je de oppervlakte van een cirkel kunt berekenen.

Verwachte output in je console zoiets als onder:

```
Oppervlakte rechthoek: 12
Oppervlakte vierkant: 16
Oppervlakte driehoek: 6
Oppervlakte cirkel: 28.27
```

Kobo Desktop

SHOP KOBO MY BOOKS EREADER SETUP *Zie Scherp Scher...*

14.4 Oefeningen

Boek

Maak een klasse Boek en gebruik auto-properties voor de velden:

- Isbn (int)
- Title (string)
- Auteur (string)

Maak voorts een full property voor Prijs (double).

Maak een child-klasse die van Boek overerft genaamd TextBoek. Een TextBoek heeft één extra property:

- GradeLevel (int)

Maak een child-klasse die van Boek overerft genaamd KoffietafelBoek. Deze klasse heeft geen extra velden.

Voorts kunnen boeken "opgeteld" worden om als omnibus uitgebracht te worden. De titel wordt dan "Omnibus van X". waarbij X de Auteurs bevat, gescheiden met een komma. De prijs van een Omnibus is steeds de som van beide boeken gedeeld door 2. **Schrijf een static methode TelOp die**

Kobo Desktop

SHOP KOBO MY BOOKS EREADER SETUP *Zie Scherp Scher...*

twee Boek objecten als parameter aanvaardt en als returntype een nieuw Boek teruggeeft.

In beide child-klassen, override de Prijs-setter zodat:

- Bij TextBoek de prijs enkel tussen 20 en 80 kan liggen.
- Bij KoffietafelBoek de prijs enkel tussen 35 en 100 kan liggen.

Zorg ervoor dat boeken de ToString overriden zodat je boekobjecten eenvoudig via Console.WriteLine(myBoek) hun info op het scherm tonen. Ze tonen deze info als volgt: "Title - Auteur (Isbn) _ Prijs" (bv. The Shining - Stephen King (05848152) _ 50)

Maak boeken aan van de 3 klassen, toon dat de prijs niet altijd zomaar ingesteld kan worden.

Bekijk zeker de appendix indien je wilt weten hoe je de + operator kunt overriden om boeken op te tellen, zodat je geen methode nodig hebt maar gewoon kunt schrijven: Boek opgeteld = boek1 + boek2;.

Money, money, money

Kobo Desktop

SHOP KOBO MY BOOKS EREADER SETUP *Zie Scherp Scher...*

Maak enkele klassen die een bank kan gebruiken (of hergebruik je BankManager code uit hoofdstuk 9)

1. Abstracte klasse Rekening: deze bevat een methode VoegGeldToe en HaalGeldAf. Het saldo van de rekening wordt in een private variabele bijgehouden (en via de voorgaande methoden aangepast) die enkel via een read-only property kan uitgelezen worden. Voorts is er een abstracte methode BerekenRente de rente als double teruggeeft.
2. Een klasse BankRekening die een Rekening is. De rente van een BankRekening is 5% wanneer het saldo hoger is dan 100 euro, zoniet is deze 0%.
3. Een klasse SpaarRekening die een Rekening is. De rente van een SpaarRekening bedraagt steeds 2%.
4. Een klasse ProRekening die een SpaarRekening is. De ProRekening hanteert de Rente-berekening van een SpaarRekening (base.BerekenRente) maar zal per 1000 euro saldo nog eens 10 euro verhogen.

Schrijf deze klassen en toon de werking ervan in je main.

Bookmark extra

Implementeer de ToString() methode in zowel de Bookmark als de HiddenBookmark klasse. Bij bookmark moet de output bestaan uit de titel

van de site, gevolgd door de url tussen haakjes, bv:

```
Google (www.google.com)
```

Bij HiddenBookmark wordt er achteraan nog "—HIDDEN—" gezet:

```
Google (www.google.com) ---HIDDEN---
```

Zorg ervoor dat er géén dubbele code in HiddenBookmark staat (tip: base()).

Geometric figures

Maak een abstracte klasse GeometricFigure. Iedere figuur heeft een hoogte, breedte en oppervlakte. Maak autoproperties voor van Hoogte en Breedte. De Oppervlakte is een read-only property want deze wordt berekend gebaseerd op de hoogte en breedte. Deze berekening gebeurt in de methode BerekenOppervlakte: deze roep je met andere woorden aan in de getter van Oppervlakte en dat resultaat geeft de getter terug

Let er dus op dat Oppervlakte enkel een getter heeft. Een setter schrijven zou vreemde bugs geven: wat als de gebruiker de oppervlakte van de figuur zo aanpast, maar wel een andere hoogte en breedte heeft

ingesteld? Je zou dan een foute oppervlakte krijgen daar die niet berekend wordt.

Voorzie een abstracte methode BerekenOppervlakte die een int teruggeeft.

Maak 3 klassen:

- **Rechthoek**: erft over van GeometricFigure. Oppervlakte is gedefinieerd als breedte * hoogte.
- **Vierkant**: erft over van Rechthoek. Voorzie een constructor die lengte en breedte als parameter aanvaard: deze moeten gelijk zijn, indien niet zet je deze tijdens de constructie gelijk. Voorzie een 2e constructor die één parameter aanvaardt dat dan geldt als zowel de lengte als breedte. Deze klasse gebruikt de methode BerekenOppervlakte van de Rechthoek-klasse.
- **Driehoek**: erft over van GeometricFigure. Oppervlakte is gedefinieerd als breedte*hoogte/2.

Maak een applicatie waarin je de werking van deze klassen aantoont.

Dierentuin

Maak een console-applicatie waarin je een zelfverzonnen abstract klasse

Dier in een List<Dier> kunt plaatsen. Ieder dier heeft een gewicht en een methode Zegt (die abstract is) die het geluid van het dier in kwestie op het scherm zal tonen. Maak enkele childklassen die overerven van Dier en uiteraard de Zegt methode overriden.

Plaats enkele dieren in de nieuw aangemaakt lijst.
Vervolgens verschijnt er een menu Wanneer de gebruiker 'q' kiest stopt het programma. Het keuze menu heeft volgende opties:

1. **Dier verwijderen** , gevolgd door de gebruiker die invoert het hoeveelste dier weg moet uit de List.
2. **Diergewicht gemiddelde**: het gemiddelde van alle dieren hun gewicht wordt getoond
3. **Dier praten**: alle dieren hun Zegt() methode wordt aangeroepen en via WriteLine getoond
4. **Opnieuw beginnen**: de List wordt leeggemaakt en het programma zal terug van voor af aan beginnen. Waarbij de lijst terug gevuld wordt met enkele start dieren.

Probeer zo modulair mogelijk te werken.

15 Opdracht 15 - Zie Scherp Scherper H15

We gaan verder met de online tutorial *Zie Scherp Scherper* en wel met *H15 Compositie en aggregatie*.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_15** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP, ZieScherper_09 tot en met ZieScherper_14, en het nieuwe project **ZieScherper_15**:

- > OOP (solution)
  - > OOP (project)
  - > ZieScherper_09 (project)
  - > ZieScherper_10 (project)
  - > ...
  - > **ZieScherper_15** (project)

Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_15**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

15.1 Stap 1 - Compositie

Lees het eerste deel van dit hoofdstuk genaamd *Compositie*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h15-compositie-en-aggregatie/0_compositie_intro

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Neem de genoemde code van de classes **PC** en **HardeSchijf** op in je C# project in een nieuwe class file genaamd PC.cs, inclusief de instantievariabele van het type HardeSchijf in de PC class. Hernoem de instantievariabele naar **cHardeSchijf1**, en voeg een tweede en derde toe van hetzelfde type: **cHardeSchijf2**en **cHardeSchijf3.**
2. Instantieer de eerste variabele op de zoals beschreven bij het kopje *Manier 1*, de tweede zoals beschreven bij het kopje *Manier 2*, en de derde zoals beschreven bij het kopje *Manier 3*.

   Maak in de Main() method van je Program class een object aan van de PC class, waarbij je zorgt dat alle drie genoemde manieren van het instantiëren gebruikt worden.
3. Voeg de genoemde autoproperty **MaxCapacity** van het type **int** toe aan je HardeSchijf class, en neem de **ToString()** *override method* op in je PC class. Pas deze method aan zodat deze de informatie over alle drie de harde schijven naar de console schrijft. Verwachte output in je console zoiets als onder:

   ```
   Deze PC is een Intel i9.
   > Capaciteit HD1: 128 Gb
   > Er is geen HD2 aanwezig
   > Capaciteit HD3: 256 Gb
   ```
4. Neem de genoemde code van de classes **Boek** en **Pagina** op in je C# project in een nieuwe class file genaamd Boek.cs, met in de Boek class de property **AllePaginas** van het List type (en niet van het tevens genoemde array type).

   Maak in de Main() method van je Program class objecten aan van beide classes, en roep de InsertPagina() method aan (en niet de List.Insert() method).

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 15.1**.

15.2 Stap 2 - this keyword

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht15_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *this keyword*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h15-compositie-en-aggregatie/this

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de genoemde code van de classes **Werknemer** en **Management** op in je C# project in een nieuwe class file genaamd werknemer.cs, en neem de instantiatie van de Werknemer class en bijbehorende method aanroep op in de Main() method.
2. Voeg een foutconditie afhandeling toe bij de controle of een werknemer wel/niet mag promoveren, en bouw en run je project. Zorg in de instantiatie van de Werknemer class ervoor dat je beide condities test.

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 15.2**.

15.3 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Oefening 'Politiek'.
   Classes **Land**, **Minister** en **President** in class file Land.cs, test method **Opdracht15_3a()**, commit **Opdracht 15.3a**.

Verwachte output in je console zoiets als onder:

```
Gaat niet. Dit land heeft al een regering.
Weer een jaar verder (1 jaar verstreken)
Weer een jaar verder (2 jaar verstreken)
Weer een jaar verder (3 jaar verstreken)
Weer een jaar verder (4 jaar verstreken)
Regering is gedaan (de jaren zijn op)
```

b) Oefening 'Moederbord'.

Class **Moederbord**, (parent class) **PCComponent**, (child classes) **CPUSlot**, **AGPSlot** en **RAM**; allen in class file Moederbord.cs. Test method **Opdracht15_3b()** en commit **Opdracht 15.3b**.

De benoemde '2 (of 4) RAM-slots' stop je in een List<RAM> type, en aan de constructor van de Moederbord class kun je het aantal RAM sloten meegeven.

Verwachte output in je console zoiets als onder:

```
Je hebt een CPU van het merk IntelCorei9_9900K.
Je hebt een AGP kaart van het merk GeForceRTX2080.
Je hebt nog 3 vrije RAM sloten.
```

15.6 Oefeningen

Politiek

Maak een programma om de politieke situatie van een land te simuleren.

Maak volgende klassen:

- Land
- Minister
- President

Minister

Een Minister heeft geen speciale eigenschappen. Enkel een autoproperty om de Naam van de minister in bij te houden

President

Een President is een Minister maar met 1 extra property met private setter: hij heeft een teller die start op 4 alsook een methode JaarVerderdie deze teller met 1 iedere aanroep verlaagt.

Land

- Een Land heeft 0 of 1 President (of koning, dictator kies zelf).
- Een Land heeft 0 of 1 "eerste minister" (m.b.v. een eersteMinister instantievariabele).
- Een Land heeft 0 tot 4 ministers (via een List<Minister>).

Al deze compositieobjecten zijn private.
Een land heeft volgende publieke methoden:

MaakRegering

Deze methode aanvaardt volgende parameters:

1. 1 President object die aan de private president variabele wordt toegekend.
2. Een List<Minister> object waarin tussen de 1 tot 5 ministers in staan: de eerste minister in de lijst wordt toegewezen aan de eersteMinister instantievariabele. De overige ministers in de lijst worden aan de private lijst van ministers toegewezen.

Deze methode zal enkel iets doen indien er geen president in het land is (null). Indien er reeds een regering is dan zal er een foutboodschap verschijnen.

JaarVerder

Deze methode aanroepen zal de JaarVerder aanroepen op de president indien deze er is (en dus niet null is). Deze methode controleert ook of de teller van de president na deze aanroep op 0 staat. Als dat het geval is dan worden alle ministers en president in het land op null gezet.

Eindfase

Controleer je klasse Land door enkele ministers en een president te maken en deze in een object van het type Land via MaakRegering door te geven. Test dan wat er gebeurt indien je enkele malen JaarVerder op het land aanroept.

Moederbord

Maak een klasse Moederbord die een, je raadt het nooit, moederbord van één computer voorstelt. Kies een van de vele moederborden die je online vindt en bekijk uit welke delen een moederbord bestaat (*heeft een*).

Maak voor ieder deel een aparte klasse. Voorzie vervolgens via compositie de nodige objecten in je moederbord. Denk er aan dat je bijvoorbeeld 2 (of 4) RAM-slots hebt en dus hier ofwel een array moet voorzien van het type List<RAM>, oftewel twee aparte delen RAMSlot1 en RAMSlot2.

Maak een methode TestMoederbord in de klasse Moederbord. Wanneer je deze

aanroept zal deze weergeven welke onderdelen nog leeg zijn (==null).

Voorzie een default constructor die alle datavelden initialiseert.

Iedere module moet via een property langs buiten ingesteld worden (beeld je in dat je effectief een moederbord in elkaar knutselt):

```
Moederbord Z390E_GAMING = new Moederbord();
Z390E_GAMING.AGP= new AGPSlot("GeForceRTX2080");
Z390E_GAMING.CPU= new CPUSlot("IntelCorei9_9900K");
//enz.
```

Kan je zelf een computer samenstellen door enkele objecten van verschillende types aan te maken en deze aan je moederbord-object toe te wijzen?

16 Opdracht 16 - Zie Scherp Scherper H16

We gaan verder met de online tutorial *Zie Scherp Scherper* en wel met *H16 Polymorfisme*.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_16** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP, ZieScherper_09 tot en met ZieScherper_15, en het nieuwe project **ZieScherper_16**:

- OOP (solution)
  - OOP (project)
  - ZieScherper_09 (project)
  - ZieScherper_10 (project)
  - ...
  - **ZieScherper_16** (project)

Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_16**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

16.1 Stap 1 - Polymorfisme

Lees het eerste deel van dit hoofdstuk genaamd *Polymorfisme*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h16-polymorfisme/11_polymo_intro

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Zorg ervoor dat je in de Program class m.b.v. een *using* statement de **Dier** class uit het ZieScherper_14 project kan gebruiken. Als je niet meer weet hoe dat moet kijk dan even in je code van Opdracht 14 (paragraaf 14.1), waarin je hetzelfde hebt moeten doen om de 'Student' class (uit het ZieScherper_10 project) te kunnen gebruiken.
2. Voeg de beschreven class **Varken** toe aan je Dier class, en los de foutmelding op die je nu krijgt op door (automatisch m.b.v. de IDE!) de ontbrekende *override property* **MaxLeeftijd** aan de nieuwe class toe te voegen.
   Maak in je Main() method instanties van de alle Dier *child classes* en roep op de objecten de **MaakGeluid()** method aan.
3. Voeg de genoemde code in het kopje 'Arrays en polymorfisme' aan je Main() method, en voeg ook een object van het type Wolf toe aan de array. Bouw en run je project en test of je dezelfde output krijgt als in de vorige stap.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand (2x het geluid van alle dieren):

```
Oinkoink
Hinnikhinnik
Geluid van een wolf

Oinkoink
Hinnikhinnik
Geluid van een wolf
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 16.1**.

16.2 Stap 2 - Polymorfisme in de praktijk

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht16_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Polymorfisme in de praktijk*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h16-polymorfisme/polypraktijd

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de beschreven code van de class **EersteMinister** en van alle genoemde **Minister** classes op in je C# project in een nieuwe class file genaamd Minister.cs, en gebruik hiervoor de versie die wordt beschreven als de 'goede manier'.
   Dus met een *abstract parent class* Minister en voor elk type minister een afgeleide child class. En zorg voor een override van de method **Adviseer()** voor elk type minister.
2. De code uit de tutorial is niet foutloos en ook niet helemaal volledig, dus probeer de fouten op te lossen en vul de code aan daar waar nodig. Zorg in ieder geval dat de nieuwe class goed bouwt voordat je verder gaat.
3. Maak in je Main() method een instantie van de EersteMinister class en roep op het object de **Regeer()** method aan. Zorg ervoor dat je in de minister classes d.m.v. *console writelines* toont welke type minister er advies geeft, én welk advies er wordt gegeven.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
De MinisterVanMilieu Jansens adviseert:
> Verhoog bossubsidies
> Open onderzoek
> Contacteer Greenpeace
De MinisterVanBZ Ganzeweel adviseert:
> Vervang ambassadeur
> Roep troepenmacht terug
> Verhoog risicozone aan grens
De MinisterVanEconomie VanCent adviseert:
> Injecteer geld in markt
> Verlaag werkloosheidspremie
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 16.2**.

16.3 Stap 3 - 'Is' en 'As' keywords

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht16_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Is en As keywords:*
https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h16-polymorfisme/1_isas

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Geef in de EersteMinister class de auto property **AlleMinisters** de visibility **public**, indien je deze in de vorige stap als private had gedefinieerd (wat op dat moment voldoende, en dus beter was).
2. Loop in je Main() method, door gebruik te maken van deze auto property, door de lijst van alle ministers en gebruik het ***is*** *keyword* om te bepalen wie de 'MinisterVanBZ' is.
3. Maak in je Main() method een instantie van de **TextBoek** class, een child class uit het ZieScherper_14 project die je al hebt ge-include in je huidige project m.b.v. een using statement.
4. Maak vervolgens een instantie van de **Boek** class, een parent class uit hetzelfde ZieScherper_14 project, en geef dit object dezelfde waarde als het TextBoek object door gebruik te maken van het ***as*** *keyword*, oftewel d.m.v. type casting.
   Check ook of de conversie is gelukt, en indien het geval print bijvoorbeeld de boektitel naar de console.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
Minister Jansens is niet de MinisterVanBZ
Minister Ganzeweel is wel de MinisterVanBZ
```

```
Minister VanCent is niet de MinisterVanBZ

Boek Titel: De titel van het Boek
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 16.3**.

16.4 Stap 4 - Objecten vergelijken

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht16_3()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Objecten vergelijken:* https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h16-polymorfisme/6_equals

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. In het ZieScherper_10 project heb je een Student class geïmplementeerd die je in paragraaf 14.1 hebt uitgebreid met een *override* versie van de **Equals()** method. Deze heb je in Main() method van het ZieScherper_14 project vervolgens gebruikt om twee student objecten met elkaar te vergelijken. Bekijk de code van de method en de aanroep nogmaals.
2. Verander in de Main() method de aanroep van de Equals() method door een object van een andere klasse als parameter mee te geven, bv de **Mens** class, ook uit het ZieScherper_10 project. In pseudo code wordt het dus zoiets als:

   ```
   if <Student object>.Equals(<Mens object>)
   then ...
   else ...
   ```

   Analyseer het run-time probleem wat er ontstaat. Je zou een exception moeten krijgen in de Equals() method omdat de type cast niet meer werkt.

3. Verbeter, aan de hand van de beschreven code uit de tutorial, de Equals() method zodat de exception niet meer optreedt. Test daarna of het vergelijken van twee student objecten ook nog steeds werkt.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
Object met inhoud 'Pietje (2000)' is niet gelijk aan
'ZieScherper_10.Mens'
Object met inhoud 'Pietje (2000)' is niet gelijk aan 'Pietje (2001)'
Object met inhoud 'Pietje (2000)' is gelijk aan 'Pietje (2000)'
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 16.4**.

16.5 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Oefening 'Een eigen huis'.

   Classes **Huis**, **Kamer**, **BadKamer**, **Gang** en **Salon** in class file huis.cs, test method **Opdracht16_5a()**, commit **Opdracht 16.5a**.

   Implementeer ook overrides van de ToString() method voor de Kamer en Huis classes zodat de console output overeenkomt met zoiets als onderstaand:

```
Kamer: 400
Salon: 500
Gang: 200
Badkamer: 500
De totale huisprijs: 1600
```

16.6 Oefeningen

Een eigen huis

Basis

Gebruik compositie, polymorfisme én overerving om een klasse Huis te voorzien van enkele kamers, waarbij iedere kamer een klasse op zich is (denk aan BadKamer, Salon, etc). Alle kamers erven over van de klasse Kamer. Iedere kamer heeft een oppervlakte (in vierkante meter), een naam en prijs. Standaard is de prijs van een kamer 400 euro, maar mogelijke child-klassen zullen deze property soms overriden:

Maak minstens volgende klassen:

- Badkamer: kost 500 euro.
- Gang: kost 10 euro per vierkante meter dat deze kamer groot is.
- Salon: kost 300 euro indien er geen schouw aanwezig is (via bool bijhouden) anders 500 euro.

De klasse Huis heeft een array van kamers. De klasse heeft ook een methode BerekenPrijs die de totale prijs van het huis berekent gebaseerd op

de prijzen van iedere kamer in de array.

Test je klasse door enkele huizen te maken en er enkele kamers in te plaatsen (bepaal zelf hoe je de kamers aan het huis toevoegt: via een methode, constructor, etc) en vervolgens de prijs ervan te tonen.

Mapmaker

Voorzie een `Teken` methode die een huis kan tekenen, gebruikmakend van de `Teken`-methoden van de kamers. Hiervoor dien je een `X` en `Y` coördinaat per , alsook lengtes en breedter per huis én kamer te hebben zodat je deze op de juiste plekken op het scherm kan plaatsen.

Kan je ervoor zorgen dat een architect nieuwe kamers kan toevoegen en verwijderen?

17 Opdracht 17 - Zie Scherp Scherper H17

We gaan verder met de online tutorial *Zie Scherp Scherper* en wel met *H17 Interfaces*, tevens het laatste hoofdstuk van de tutorial.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_17** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP, ZieScherper_09 tot en met ZieScherper_16, en het nieuwe project **ZieScherper_17**:

- OOP (solution)
  - OOP (project)
  - ZieScherper_09 (project)
  - ZieScherper_10 (project)
  - ...
  - **ZieScherper_17** (project)

Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_17**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

17.1 Stap 1 - Interface intro

Lees het eerste deel van dit hoofdstuk genaamd *Interface intro*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h17-interfaces/1_interface_intro

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Neem de beschreven code van de interface **ISuperHeld** én van de op deze interface gebaseerde class **Zorro** op in je C# project, beide in een aparte class file met dezelfde file name als de class zelf. Controleer of je project goed bouwt.
2. Haal in de Zorro class één van de geïmplementeerde interface properties of methods weg (je project zal niet meer compileren), en probeer met de in de tutorial beschreven tip de weggehaalde property of method met behulp van de IDE **automatisch** te (her)genereren. Controleer of je project weer goed bouwt, en indien het geval kun je de code weer in de oude staat terugbrengen.
3. Maak in je Main() method een instantie van de Zorro class, en bepaal m.b.v. het ***is*** *keyword* of het de ISuperHeld interface implementeert. Print het resultaat naar de console.
4. Voeg aan de file waarin je Zorro class staat ook een nieuwe (lege) class **Man** toe. Pas de definitie van de Zorro class aan zodat deze overerft van de Man class, maar tegelijk ook nog steeds de ISuperHeld interface implementeert.
   Bepaal in je Main() method m.b.v. het ***is*** *keyword* of de Zorro class overerft van de Man class, en print het resultaat naar de console.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
De class Zorro implementeert de ISuperHeld interface
De class Zorro erft over van de parent class Man
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 17.1**.

17.2 Stap 2 - Interfaces voorbeeld

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht17_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Interfaces voorbeeld met president*:

https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h17-interfaces/presidentinterfaces

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Zorg ervoor dat je in de Program class m.b.v. een *using* statement de **Minister** class uit het ZieScherper_16 project kan gebruiken. Als je niet meer weet hoe dat moet kijk dan even in je code van Opdracht 16 (paragraaf 16.1), waarin je hetzelfde hebt moeten doen om de 'Dier' class (uit het ZieScherper_14 project) te kunnen gebruiken.
2. Bestudeer in het ZieScherper_16 project nogmaals de code van de Minister classes die je hebt geïmplementeerd in de class file Minister.cs. Deze classes gaan we aanpassen door gebruik te maken van een interface.
3. Neem de beschreven code van de interface **IMinister** én van de op deze interface gebaseerde class **Ceo** op in je ZieScherper_16 project (dus niet in het huidige ZieScherper_17 project), en beide in een aparte class file met dezelfde file name als de class zelf.
   De Ceo class krijgt dus twee eigen (interface-onafhankelijke) methods, en de (verplichte) interface method **Adviseer()**. Controleer of je ZieScherper_16 project nog goed bouwt.
4. Pas, zoals uitgelegd in de tutorial, de **EersteMinister** class aan door op de juiste plekken gebruik te maken van de *interface class* IMinister i.p.v. de *abstract parent class* Minister. Controleer of je ZieScherper_16 project nog steeds goed bouwt.
5. Voeg in de constructor van de EersteMinister ook een Ceo class object toe aan de lijst met ministers, wat nu mogelijk is aangezien de Ceo ook de IMinister interface implementeert.
6. Voer in het ZieScherper_16 project opnieuw de test method **Opdracht16_2()** uit. Hierin wordt een instantie van de EersteMinister class aangemaakt, en vervolgens de **Regeer()** method aangeroepen.

Analyseer de output naar de console. D.m.v. de bestaande *console writelines* wordt nog steeds getoond welke type minister er advies geeft, én welk advies er wordt gegeven, waarbij het advies van de Ceo nu als nieuw advies is toegevoegd.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand, met in het rood het nieuwe advies afkomstig van de Ceo:

```
De MinisterVanMilieu Jansens adviseert:
> Verhoog bossubsidies
> Open onderzoek
> Contacteer Greenpeace
De MinisterVanBZ Ganzeweel adviseert:
> Vervang ambassadeur
> Roep troepenmacht terug
> Verhoog risicozone aan grens
De MinisterVanEconomie VanCent adviseert:
> Injecteer geld in markt
> Verlaag werkloosheidspremie
De Ceo Pietersen adviseert:
> Vrijhandel is essentieel!
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 17.2**.

17.3 Stap 3 - Interfaces in de praktijk

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht17_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Interfaces in de praktijk:* https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h17-interfaces/2_interfacesinpraktijk

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de beschreven code van de class **Land** class op in je C# project in een nieuwe class file genaamd Land.cs, en zet de bijbehorende code die een array met landen maakt in je Main() method.
2. Voeg ook het genoemde statement toe die de lijst met landen sorteert. Bouw en run je project en controleer of er inderdaad een exception optreedt.
3. Los het probleem op door de **IComparable** interface in de Land class te implementeren. Zorg voor de volgende volgorde in sortering: eerst op **oppervlakte**, daarna op aantal **inwoners**, en tenslotte op de **naam** van het land.
4. Test je implementatie door zowel voor als na de sortering de lijst met landen te printen, met al zijn properties, en check of je sorteer-algoritme goed werkt.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand:

```
Lijst voor sortering:
Naam: België - Oppervlakte: 5 - Inwoners: 2000
Naam: Frankrijk - Oppervlakte: 7 - Inwoners: 2500
Naam: Nederland - Oppervlakte: 6 - Inwoners: 1800
Naam: Engeland - Oppervlakte: 6 - Inwoners: 1700
Naam: Duitsland - Oppervlakte: 6 - Inwoners: 1700

Lijst na sortering:
Naam: België - Oppervlakte: 5 - Inwoners: 2000
Naam: Duitsland - Oppervlakte: 6 - Inwoners: 1700
Naam: Engeland - Oppervlakte: 6 - Inwoners: 1700
Naam: Nederland - Oppervlakte: 6 - Inwoners: 1800
Naam: Frankrijk - Oppervlakte: 7 - Inwoners: 2500
```

Indien alle testen succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 17.3**.

17.4 Stap 4 - Interfaces en polymorfisme

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht17_3()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Interfaces en polymorfisme:* https://apwt.gitbook.io/zie-scherp-scherper/object-oriented-programming/h17-interfaces/2_polymorfisme_interfaces

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de beschreven code van de parent class **Mens** en de child classes **Leraar**, **Student**, **Advocaat** en **Politieker** op in je C# project in een nieuwe class file genaamd Mens.cs.
2. Neem de code van de interface **IVloeker** op in je C# project, in een aparte class file met dezelfde file name als de class zelf. Controleer of je project nog goed bouwt.
3. Maak in je Main() method de genoemde array (of beter: een List<>) van **Mens** objecten aan en loop er met een for (of: foreach) loop doorheen. Roep voor elk object de **Spreek()** method aan.
4. Implementeer beide beschreven oplossingen die het probleem oplost om in de array/list van mensen enkel de vloekende mensen laten vloeken, oftewel eerst m.b.v. het ***is*** *keyword* en daarna m.b.v. het ***as*** *keyword*.

De verwachte output in je console zou na alle stappen zoiets moeten zijn als onderstaand (3x het resultaat van de for/foreach):

```
Hoi!
Hoi!
Hoi!
Hoi!

Hoi!
Godvermiljaardedju, zei de politieker
Hoi!
SHIIIIT, zei de advocaat

Hoi!
Godvermiljaardedju, zei de politieker
Hoi!
```

```
SHIIIIT, zei de advocaat
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 17.4**.

17.5 Laatste stap - Oefeningen

Maak de onderstaande oefeningen waarvan je de tekst in de screenshots kunt vinden. Voor elke oefening maak je een nieuwe class file aan die je vervolgens test in je Main() method. Indien alle testen van een oefening succesvol zijn verlopen, *commit* en *push* je de code naar je GitHub repo.

Verplaats voor elke nieuwe oefening de code uit de Main() naar een nieuwe *private static void* test method zodat je weer met een lege Main() start voor de volgende oefening.

a) Implementeer de benodigde **CompareTo()** method in je bestaande **ComputerOnderdeel** class die je in H12 hebt gemaakt (d.m.v. een project reference), dus NIET kopiëren.
In de test method waar je deze class test, ook in het ZieScherper_12 project dus, sorteer je de bestaande lijst met computer onderdelen op prijs nadat je alle Verwijder_<> methods hebt aangeroepen (oftewel sorteer de 'overgebleven onderdelen').
Test je implementatie door zowel voor als na de sortering de lijst met onderdelen te printen. Verwachte output in je console zoiets als onder.

```
Overgebleven onderdelen (ongesorteerd):
Prijs: 475, ID: 269, InDoos: False
Prijs: 357, ID: 653, InDoos: False
Prijs: 104, ID: 419, InDoos: False
Prijs: 514, ID: 387, InDoos: False
Prijs: 380, ID: 772, InDoos: False

Overgebleven onderdelen (gesorteerd):
Prijs: 104, ID: 419, InDoos: False
Prijs: 357, ID: 653, InDoos: False
```

```
Prijs: 380, ID: 772, InDoos: False
Prijs: 475, ID: 269, InDoos: False
Prijs: 514, ID: 387, InDoos: False
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 17.5a**.

b) Oefening 'Carbon Footprint'.

Classes **Huis**, **Fabriek**, **Auto**, en **Plant** in class file CarbonFootPrint.cs; class **ICarbonFootPrint** in een aparte class file met dezelfde file name als de class zelf; test method **Opdracht17_5b()**, commit **Opdracht 17.5b**.

De tekst in de oefening is er niet helemaal duidelijk in, maar wij kiezen ervoor (om redenen die je hopelijk duidelijk worden) om de class Plant NIET aan de interface te laten voldoen.

Verwachte output in je console zoiets als onder:

```
Huis[1]: 1800
Fabriek[2]: 990
Auto[3]: 18
Huis[5]: 1000
Fabriek[6]: 2500
Auto[7]: 17
De gemiddelde footprint: 790
Vervuiler Fabriek op positie 6 heeft grootste footprint: 2500
```

17.7 Oefeningen

ComputerOnderdelen sorteren

Haal de Computerwinkel oefening uit hoofdstuk 12 boven. Zorg ervoor dat je elementen van het type `ComputerOnderdeel` kunnen gesorteerd worden op hun Prijs. Voorzie een nieuw menuitem "Sorteer onderdelen" die het gesorteerde resultaat toont.

Carbon Footprint

Maak 4 klassen:

- Huis
- Fabriek
- Auto
- Plant

We maken een programma om de fictieve ecologische voetafdruk van bepaalde objecten te berekenen

Maak een interface `ICarbonFootPrint` die volgende zaken heeft:

- 1 methode BerekenFootprint die een int teruggeeft en geen parameters nodig heeft.
- 1 methode VerlaagFootprint die niets teruggeeft en geen parameters nodig heeft.

Breid de volgende klassen met de interface uit:

- De carbon footprint van een Huis is gebaseerd op het volume van het huis in kubieke meter maal 10.
- De carbon footprint van een Fabriek is gebaseerd op het aantal werknemers maal 100.
- De carbon footprint van een Auto is gebaseerd op het merk.
- De carbon footprint van een Plant is altijd 0.

Het verlagen van de footprint in iedere klasse verzin je zelf (door bijvoorbeeld bij het huis de factor 10 met 1 te verlagen), maar merk op dat deze nooit negatief mag zijn.

Zorg ervoor dat van iedere klasse de footprint kan bevraagd worden (maak/verzin dus de nodige properties per klasse om dit te berekenen).

Plaats van iedere klasse 2 objecten in een gemeenschappelijke lijst en zorg ervoor dat:

- De footprint van alle objecten getoond wordt (planten worden

overgeslagen).

- De gemiddelde footprint van alle objecten berekend wordt.
- Toont welk object de hoogste footprint heeft.
- Van alle objecten de footprint kan verlaagd worden.

18 Opdracht 18 - Zie Scherp Scherper Appendix [Optioneel]

In dit hoofdstuk behandelen we de wat meer geavanceerde OOP/C# concepten, gedeeltelijk afkomstig uit de appendix van de online tutorial *Zie Scherp Scherper*, maar ook afkomstig van ander internetbronnen.

GitHub repo en MS Visual Studio

We blijven werken in dezelfde repo en maken in MS Visual Studio in de bestaande **OOP** solution een nieuw C# project met de naam **ZieScherper_18** gebaseerd op de template 'Console Application C#' (rechter muis op solutionnaam > Add > New Project).

Je hebt nu dus de volgende projecten in je solution: de bestaande projecten OOP, ZieScherper_09 tot en met ZieScherper_17, en het nieuwe project **ZieScherper_18**:

- OOP (solution)
  - OOP (project)
  - ZieScherper_09 (project)
  - ZieScherper_10 (project)
  - ...
  - **ZieScherper_18** (project)

Update daarna, indien nodig, de code van *Program.cs* zodat die begint met een *namespace* genaamd **ZieScherper_18**, met daarin een class genaamd **Program**, met daarin een *public static void **Main()** method*. Zie ook de uitleg in paragraaf 1.2 met waar je waarschijnlijk dezelfde aanpassing hebt moeten doen.

Maak, in Visual Studio, in het nieuwe project een nieuwe folder **classes** aan waarin je nieuwe class files komen te staan (rechter muis op *projectnaam* > Add > New Folder).

18.1 Stap 1 - Out en Ref parameters

Lees het eerste deel van dit hoofdstuk genaamd *Out en Ref parameters*:

https://apwt.gitbook.io/zie-scherp-scherper/programming-principles-appendix/prostuff/2_outenref

Voer **nadat je alles hebt gelezen** de volgende stappen uit:

1. Zorg dat de beschreven code van de method **VerhoogWaarde()** in je Program class terecht komt, dus niet alleen de method zelf, maar ook de aanroep ervan. Maak van de method een *private static void* method, zodat je deze in je *Main()* kan aanroepen.
2. Zorg dat de beschreven code van de method **GeefWaarde()** in je Program class terecht komt, dus niet alleen de method zelf, maar ook de aanroep ervan.

De verwachte output in je console zou na deze stappen zoiets moeten zijn als onderstaand:

```
Waarde voor methode met 'ref': 1
Waarde na methode met 'ref': 2
Waarde na methode met 'out': 5
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 18.1**.

18.2 Stap 2 - TryParse

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht18_1()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Foute invoer van de gebruiker opvangen m.b.v. TryParse*:

https://apwt.gitbook.io/zie-scherp-scherper/programming-principles-appendix/prostuff/2_outenref#foute-invoer-van-de-gebruiker-opvangen-m.b.v.-tryparse

Voer nadat je alles hebt gelezen de volgende stappen uit:

1. Neem de beschreven code die de leeftijd van de gebruiker opvraagt op in de Main() method van je Program class. Run je project een aantal maal en voer zowel geldige als ongeldige leeftijden in.

2. Los het exception probleem op door gebruik te maken van de **TryParse()** method. Run je project nogmaals een aantal maal en voer wederom zowel geldige als ongeldige leeftijden in. Controleer of je in alle gevallen je applicatie kunt runnen zonder dat het programma crasht.

De verwachte output in je console zou na deze stappen zoiets moeten zijn als onderstaand:

```
Geef je leeftijd:
18
Over 10 jaar ben je 28 jaar oud

Geef je leeftijd:
abc
Geen geldige invoer gegeven!
```

Indien het geval, *commit* en *push* je de code naar je GitHub repo met als message: **Opdracht 18.2**.

18.3 Stap 3 - Operator overloading (TBD)

Maak in je Program class een nieuwe *private static void* method genaamd **Opdracht18_2()** en verplaats alle code die nu in je Main() method staat hier naartoe, zodat je Main() weer leeg is en je code uit de vorige stap niet verloren gaat.

Lees het volgend deel van dit hoofdstuk genaamd *Operator overloading:*
https://apwt.gitbook.io/zie-scherp-scherper/oop-appendix/nice-to-know-stuff/8_operatoroverloading

Voer nadat je alles hebt gelezen de volgende stappen uit:
<To Be Done>

Bijlage A - Een project reference toevoegen

Op verschillende plekken in de lesstof word je gevraagd om een zogenaamde *project reference* toe te voegen aan je huidige project (bv 'ZieScherper_11'). Bijvoorbeeld als je gebruikt wil maken van een class (bv Student) uit een ander project (bv 'ZieScherper_10').

Hieronder worden de (twee) stappen beschreven die nodig zijn om dat te doen, en daaronder een screenshot van het bovenstaande voorbeeld.

Add a Project Reference

1. **Right-click the project** where you want to use the class in Solution Explorer.
2. Choose `**Add**` -> `**Project Reference...**`.
3. In the Reference Manager window, under the `**Projects**` -> `**Solution**` tab, find the project that contains the class you want to use.
4. **Check the box** next to that project.
5. Click `**OK**` to add the project reference.

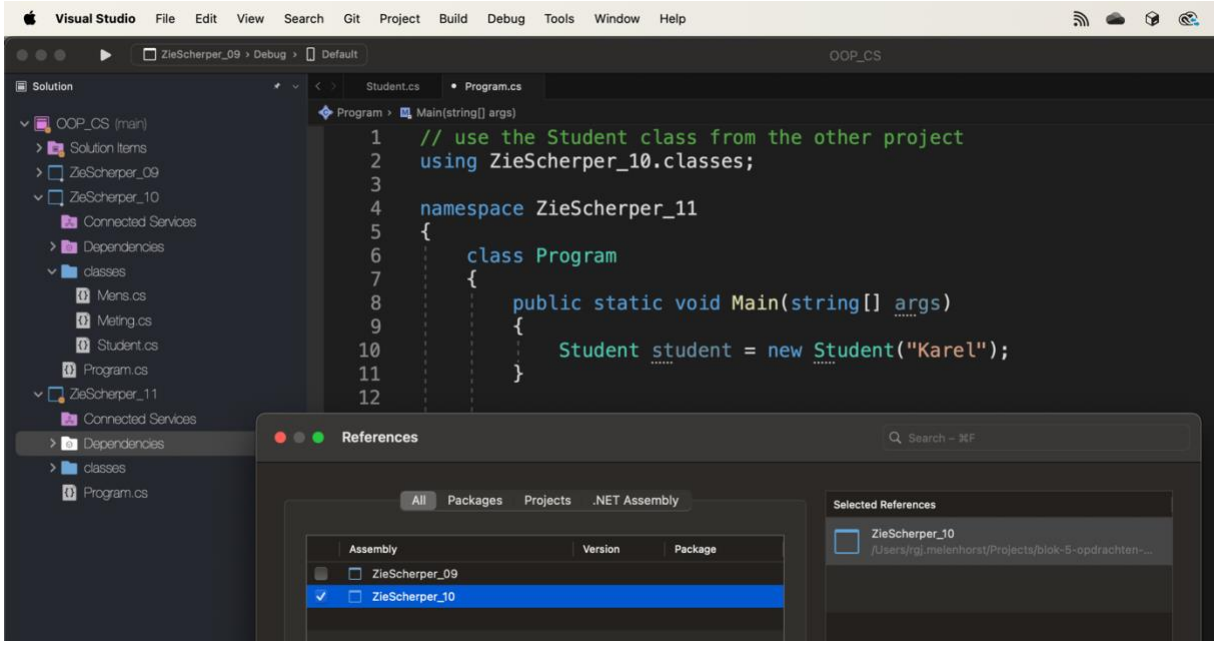
Use the Namespace

After adding the project reference, include the namespace of the class at the top of your code file with the `**using**` statement. For instance, if your class is in the `**Utilities**` namespace:

csharp Copy code

```csharp
using Utilities;
```

Now, you can access the class as needed in your code.


Visual Studio File Edit View Search Git Project Build Debug Tools Window Help
ZieScherper_09 › Debug › Default
OOP_CS
Solution
Student.cs
Program.cs
Program › Main(string[] args)
OOP_CS (main)
Solution Items
ZieScherper_09
ZieScherper_10
Connected Services
Dependencies
classes
Mens.cs
Meting.cs
Student.cs
Program.cs
ZieScherper_11
Connected Services
Dependencies
classes
Program.cs
// use the Student class from the other project
using ZieScherper_10.classes;
namespace ZieScherper_11
{
class Program
{
public static void Main(string[] args)
{
Student student = new Student("Karel");
}
References
Search – ⌘F
All Packages Projects .NET Assembly
Assembly Version Package
ZieScherper_09
ZieScherper_10
Selected References
ZieScherper_10
/Users/rgj.melenhorst/Projects/blok-5-opdrachten-...