

Pre Assessment Course

FHICT software docenten

software startsemester FHICT

April 11, 2018

master @ b40f2ce*

Copyright 2017 by FHICT software docenten.

The contents of this book are protected under the Creative Commons Attribution-ShareAlike 3.0 Unported license.

You are **free**:

- to **Share**: to copy, distribute and transmit the work,
- to **Remix**: to adapt the work,

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page:

<http://creativecommons.org/licenses/by-sa/3.0/>

Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.



Your fair dealing and other rights are in no way affected by the above. This is a human-readable summary of the Legal Code (the full license):

<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

Contents

Illustrations	iv
I Course Info: PAC - Pre Assessment Course	
1 Inleiding en Leeswijzer	3
II ProgrammeerConcepten	
2 Hoe programmeer je met Objecten?	7
2.1 Hero tegen Monsters	7
2.2 Gezondheid	8
2.3 Attack	10
3 Graphics	13
3.1 Graphics tutorial: Inleiding	13
3.2 Opdracht	13
3.3 Uitbreiding 1(niveau 3 / 5)	19
3.4 Uitbreiding 2(niveau 4 / 5)	19
4 Welke classes zal ik maken?	21
5 Properties	23
6 Private	25
6.1 Private	25
6.2 Methods	25
6.3 Properties	25
6.4 Waarom?	26
6.5 Encapsulation	26
6.6 External References	27
7 Scheiding tussen GUI en Domain	29
7.1 Domein	29
7.2 Principes	29
7.3 Illustraties	30
8 Text File Handling	33
8.1 Terminologie	33
8.2 File handling klassen	33
8.3 Andere bronnen	33
9 Exception Handling	35
10 Enum	37
10.1 Definitie van Enum	37
10.2 Voorbeelden	37

11	Override ToString()	39
12	Class Diagram	41
III Challenges		
13	Object Oriented Invul-Oefening	45
14	Challenge BattleSim	47
14.1	Inleiding	47
14.2	Opdracht	48
14.3	Extra	48
15	Challenge Vier op een rij	49
15.1	Vier op een rij	49
16	Challenge Galgje	51
17	Challenge Ontwerp Webshop Spierballetje	53
17.1	De opdracht	53
18	Challenge Platenmaatschappij	55
19	Challenge BankRekening	57
19.1	Opdracht	58
19.2	Uitbreidingen	60
20	Challenge Invaders	63
20.1	Inleiding	63
20.2	Opdracht	63
21	Challenge Auto Dagwaarde	65
22	Challenge Exception Handling	67
22.1	Opdracht 1	67
22.2	Opdracht 2	68
23	Challenge Dino Game	69
23.1	De opdracht – Dino-spel	69
24	Challenge Cube Graphic	71
24.1	The Cube	71
25	Challenge Ms PacMan	73
25.1	De opdracht	73
26	Challenge Woordenzoeker	75
27	Challenge Super Galgje	79
27.1	Super-galgje	79
28	Challenge File Handling	81
29	Challenge Iedereen kan Schilderen	85
29.1	Leerdoelen	85
29.2	Casus 1 - Iedereen kan schilderen	85
29.3	Casus 1a - Advanced painter	86
29.4	Casus 2 - Snake	86
29.5	Casus 2a - Snake Pro	86

30 External Challenges	87
31 Create your own Challenge	89
32 Challenge Uitbreiding BankRekening (UWP)	91
33 Challenge Windows Phone	93
33.1 Voorkomende leerdoelen	93
33.2 Vereiste voorkennis	93
34 Advanced Challenge Memory in Unity 3D	95
IV Bijlagen	
35 Visual Studio installeren	99
36 CodingGuidelines	105
37 Programmer Search Scheme	107
37.1 Understand	107
37.2 (Create) Plan	107
37.3 Execute Plan	107
37.4 Adjust	107
37.5 Validate Result	109
37.6 Learn/Look back	109
38 External Resources	111
38.1 Online C-sharp boeken	113
39 Programma-zip inleveren in Canvas	115
40 Handige sneltoetsen en opties in Visual Studio	117
41 Git intro and basic use	119
42 Variabelen	123
42.1 Typen variabelen	123
42.2 Variabele aanmaken (declareren)	123
42.3 Waarde aan variabele geven (toekenning of assignment)	124
42.4 Variabele aanmaken en direct een waarde geven (declare en initialize)	125
42.5 Waarden omzetten naar andere typen (convert)	125
42.6 String bewerkingen (String functies)	126
42.7 int en double bewerkingen (operatoren)	127
43 Keuzestructuren	129
43.1 if-statement	129
43.2 if ... else ... statement	130
43.3 Voorbeelden "if ..."statement en "if ... else ..."statement	130
44 Herhalingsstructuren	133
44.1 while statement	133
44.2 do while statement	133
44.3 Voorbeelden while en do while statement	134
44.4 for statement	134
44.5 Voorbeelden for statement	135
45 Methoden	137
45.1 Algemene structuur methoden	137
45.2 Belangrijkste voordeelen van het gebruik van methoden:	137
45.3 Voorbeelden Methoden	138

Illustrations

2-1	MonsterGame	7
2-2	Class Monster	8
2-3	Schematische weergave	9
2-4	Iets handigere schematische weergave	9
3-1	Paint Event	15
3-2	Teken App	18
12-1	Relation in class diagram	41
14-1	battlesim	47
16-1	galgje	51
18-1	platenmaatschappij	56
19-1	bankrekening	57
19-2	niveau	60
21-1	dagwaardeberekening	65
22-1	naamgenerator	67
23-1	dino	70
24-1	cube	71
25-1	pacman	74
25-2	pil	74
25-3	muur	74
26-1	woordenzoeker	75
28-1	Mijn Computer.	83
35-1	VSo20-components	99
35-2	Dit is Visual Studio	100
35-3	Dit is Visual Studio	100
35-4	View Code	101
35-5	password code	101
35-6	run	102
35-7	stop	102
35-8	build error	102
35-9	tikfout	103
37-1	Programmer Search Scheme	108
39-1	default project directory van Visual Studio	115

Illustrations

39-2 default project directory van Visual Studio	115
41-1 git	119
41-2 new project	119
41-3 git https	120
41-4 bash	121
41-5 git from xcode	121
41-6 git gud	122

Part I

Course Info: PAC - Pre Assessment Course

Inleiding en Leeswijzer

Dit document behoort bij de Pre-Assessment Course voor VWO-studenten die bij FHICT de Canvas Course volgen voor verkorte instroom. Informatie over de leerdoelen, structuur van de course en verwachtingen ten aanzien van de studenten is te vinden in genoemde Canvas Course.

Dit document bestaat uit een aantal *parts*. Na dit *part* met algemene informatie komen het *part* met de 'ProgrammeerConcepten' en een *part* met een collectie *Challenges*. Tot slot volgt een *part* met de 'Bijlagen' met daarin onder andere *Algemene software engineering info* zoals over bijvoorbeeld installatie van ontwikkelomgeving, het zippen van gemaakte software en handige short-cut-keys.

Losse hoofdstukken en andere versies van dit document kun je vinden op <https://github.com/coentjo/softwarelessons>.

Deze documenten zijn gecreëerd met behulp van *Pillar-markup*, zie <https://github.com/pillar-markup/pillar-documentation> voor meer info, zodat we verschillende cursussen gebruik kunnen maken van deels overlappende hoofdstukken.

Deze hoofdstukken zijn begonnen als verzameling van de lesmaterialen die we bij *FHICT* hebben voor een *introductie in object oriented programming*, in eerste instantie gericht op programmeren in C#. Het materiaal mag gedeeld worden, en bevat daarom een *Creative Commons*-licentie. Als er vragen hierover zijn horen we dat graag! (email naar C.Crombach@Fontys.nl)

Part II

ProgrammeerConcepten

CHAPTER 2

Hoe programmeer je met Objecten?

2.1 Hero tegen Monsters

Hoe groter software wordt, hoe tijdrovender het testen en onderhouden. Daarom wordt in de softwarewereld gezocht naar manieren om programma's onderhoudbaar te maken. Een van de meer succesvolle manieren is het werken met *objecten*.

Een team dat een Computer Game maakt over een *hero* die tegen *monsters* vecht zal liefst op één plek willen programmeren wat de eigenschappen van de hero zijn en ook wat een monster is en wat je aan het monster kunt vragen.

Als er 2 monsters zijn worden er 2 *objecten* gemaakt, als er 8 monsters zijn worden er 8 *objecten* gemaakt. Elk van deze objecten represeneert 1 monster. De eigenschappen en het gedrag van een monster wordt geprogrammeerd in een stuk van het programma dat we een *Class* (klasse) noemen en dat (in dit voorbeeld) de naam **Monster** krijgt.

Voorbeeld in C# (hoe je dit in Visual Studio kunt doen komt een stukje verder):

```
class Monster {  
    ...  
}
```

waarbij op de plaats van de puntjes de code voor deze *class* komt.

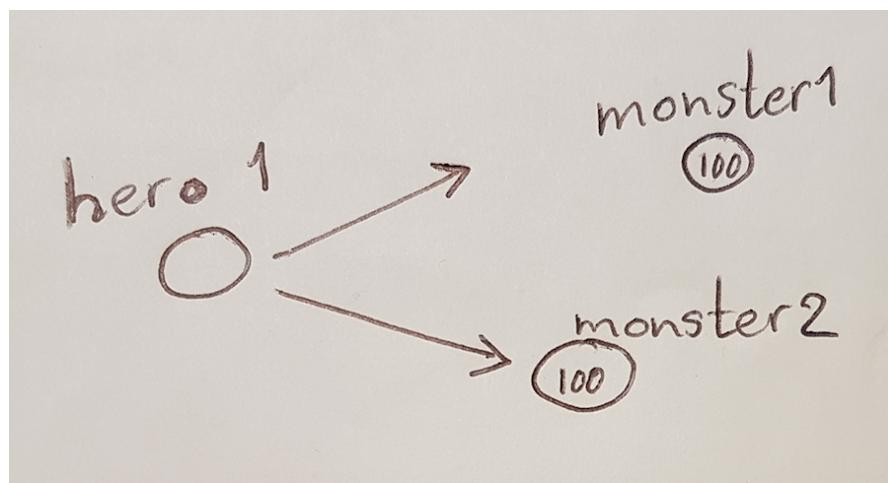


Figure 2-1 MonsterGame



Figure 2-2 Class Monster

Voorbeeld in Java:

```
[class Monster {  
    ...  
}]
```

Voorbeeld in Swift:

```
[class Monster {  
    ...  
}]
```

Inderdaad, deze voorbeelden zijn hetzelfde. Je zult merken dat er echt wel verschillen zijn hoe je in de ene of de andere taal een `class` noteert, maar in welke taal je ook zit: *welke classes je aanmaakt blijft hetzelfde!*

Note In veel programmeertalen is afgesproken dat de naam van een `class` met een hoofdletter begint.

Software Engineers bedenken in het begin van een project welke objecten er nodig zijn en daaruit volgt welke `classes` er geprogrammeerd gaan worden. Dit kun je grotendeels bedenken zonder te weten in welke taal de software gebouwd gaat worden! Je kunt dat enigszins vergelijken met het bouwen van een huis: Waar de muren, ramen en deuren komen (de structuur) kun je tot zekere hoogte bedenken en tekenen zonder te weten of het huis met bakstenen, van beton of van hout gebouwd gaat worden!

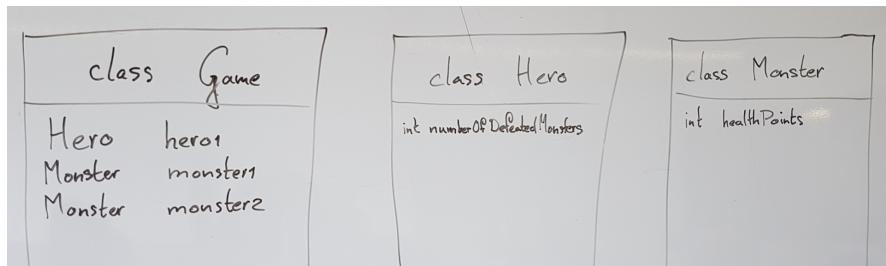
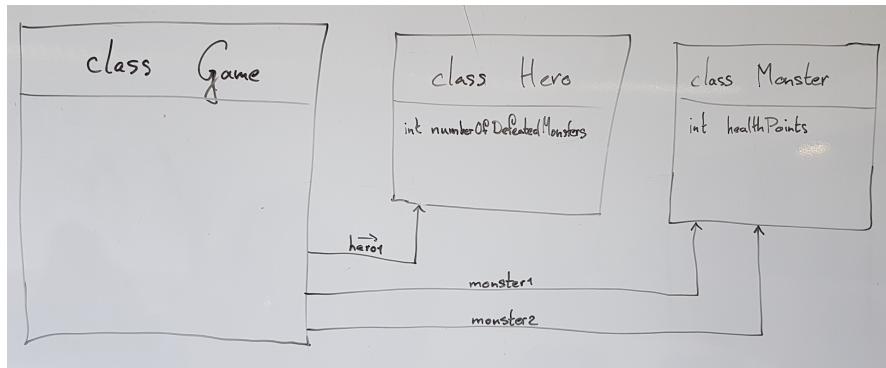
Een object kan bepaalde eigenschappen hebben. Zo zal elk Monster in eerste instantie helemaal gezond zijn. Als de *hero* hem aanvalt zal het *monster* moe worden of gewond raken en uiteindelijk wellicht bezwijken.

Hoe maak ik een class aan in Visual Studio?

Klik met rechtermuisknop op het project en kies `Add Item`, kies daarna een `class`. Onderin het scherm kun je de gewenste `class name` angeven (`file name` is `class name` met extensie `.cs`) en dan op de `OK`-knop.

2.2 Gezondheid

In dit spel kunnen we dat realiseren door het monster een getal *health* te geven: Voor een pas *aangemaakt* monster staat dit op 100, bij verwondingen wordt dit gehele getal steeds kleiner, bij 0 valt het monster verslagen neer. Een waarde als *health* die elk object van een bepaald type met zich meedraagt noemen we een *Field*. We maken hiertoe in *Monster* een veld *levenspunten* aan.

**Figure 2-3** Schematische weergave**Figure 2-4** Lets handigere schematische weergave

```

class Monster {
    int health = 100;
}

```

Hiermee is bepaald dat **elk** monster een *health* heeft. De waarde van dat getal kan per *monster object* verschillen: *Monster 1* kan nog op 100 staan terwijl *monster 2* misschien nog maar 13 over heeft.

De code die in een *class* staat wordt gedeeld met alle objecten van die *class* (meestal zeggen we: alle objecten van dat type, want een *class* is een manier om een type te definiëren). Om een object van *class Monster* aan te maken:

C# of Java:

```
[ new Monster()
```

(we zeggen dan ook wel dat er gebruik gemaakt wordt van de *new* operator)

Hiermee wordt ergens in het geheugen een object van type *Monster* aangemaakt, we hebben echter **geen** manier om naar dat object te verwijzen (*refereren*). Vergelijk het met een ballon met gas: zolang je het touwtje hebt (de referentie naar de balon) kun je bij de ballon, maar als je het touwtje loslaat kun je niet meer bij de ballon komen.

Zo'n *referentie* kunnen we opslaan in een *Field* (ook wel een variabele genoemd) en dat *Field* moet ergens in een *class* zitten: We maken hiervoor een *Game-object* aan dat de referenties naar alle *heroes* en *monsters* bevat. De code van het *Game object* komt in de *class Game* te staan.

In *class Hero* is een *Field*

```
[ int numberDefeatedMonsters = 0;
```

aangemaakt. De held wil namelijk graag dat de hele wereld weet hoeveel monsters er door hem/haar verslagen zijn.

De *Class Game* heeft referenties naar 1 *hero* en 2 *monsters* (*monster 1* en *monster 2*) en aan het ervoor vermelde type (dat zijn de *class* namen) kun je zien dat de *hero* zich gedraagt zoals in *class*

'Hero' geprogrammeerd is, terwijl de beide monsters zich gedragen volgens de code in class Monster.

2.3 Attack

Onze *hero* staat te popelen om een monster te gaan aanvallen. Hiervoor gaan we *gedrag* in de class *Hero* programmeren:

PLAATJE class Hero met method Attack(Monster monster, int damage)

Dit wil zeggen dat je naar een object van type *Hero* een *bericht* kunt sturen die *Attack* heet. Verder vertel je **welk** monster aangevallen wordt en hoeveel *schade* (*damage*) hierbij toegebracht wordt aan het monster (dus hoeveel er van de *health* punten van het monster af gaan).

Als het bericht *Attack* naar een *Held* object gestuurd wordt (in veel programmeertalen praten we niet over *een bericht sturen* maar noemen we het *een methode aanroepen*) wordt de code van die methode uitgevoerd.

De held handelt dit bericht af door een bericht *LooseHealth* te sturen naar het monster dat tussen de haakjes genoemd wordt.

Nu gaan we coderen hoe het afhandelen van het bericht er uit kan zien. We zeggen dan dat we de method (want zo'n bericht heet in veel programmeertalen een method) *Attack* in de class *Hero* programmeren

```
void Attack(Monster monster, int damage)
{
    monster.LooseHealth(damage);
}
```

ofwel: als een object van type *Hero* (want in die class staat deze code) het bericht *Attack* krijgt (met achter *Attack* tussen haakjes aangegeven **welk** monster en hoeveel *damage*) stuurt ie een bericht *LooseHealth* naar het aangegeven monster. De binnengekomen info over hoeveelheid *damage* wordt doorgegeven aan dit nieuwe bericht.

In de methode worden 2 zogenaamde parameters gebruikt, namelijk *monster* van het type *Monster* en *damage* van het type *int* (tussen haakjes te vinden na de methode naam).

Het woord *void* wil zeggen dat er geen waarde wordt teruggegeven door de methode, Er kan ook in plaats van *void* een zogenaamd *return* type staan dat aangeeft wat voor soort waarde er terug gegeven wordt.

Constructie van een object

Net zoals we bij een methode aanvullende informatie mee kunnen geven in de vorm van parameters kunnen we dat bij het aanmaken van een nieuw object ook. Hiertoe gebruiken we een *constructor*: Een constructor ziet er ongeveer uit als een methode:

```
Monster(int initialHealth)
{
    this.health = initialHealth;
}
```

Een constructor herken je als volgt:

- De constructor lijkt heel erg op een normale methode, maar...
- Er wordt geen *return*-type (of *void*) vermeld.
- De naam (*Monster* in dit geval) is gelijk aan de naam van de class.

Constructor in Visual Studio

Je kunt natuurlijk de code hierboven zelf intypen (tussen de accolades van de `class`) maar als je op die plek intypt `ctor` en dan 2x op `tab` drukt doet Visual Studio een deel van het werk voor je.

Als we nu `new Monster(125)` aanroepen vanuit code wordt er een object van type `Monster` geconstrueerd en daarvoor staat na constructie de `health`-waarde op het meegegeven getal, 125 dus in dit geval.

Bij een constructor kunnen (net als bij een *normale* methode) ook meerdere parameters meegegeven worden.

CHAPTER 3

Graphics

3.1 Graphics tutorial: Inleiding

In deze opdracht gaan we kennismaken met het zelf tekenen van figuren.

Het Graphics object

Elk object van het type *Form*, *Button*, *Panel*, *PictureBox*, etc. (ook wel *control object* genoemd) heeft een bijbehorend *Graphics* object. Dit *Graphics* object maakt het mogelijk om op de achtergrond van de control te tekenen. Het *Graphics* object van een control kun je opvragen met de method *CreateGraphics()*:

```
[Graphics graphics = CreateGraphics();
```

Na het opvragen van het *Graphics* object kun je deze gebruiken om te tekenen. De regel

```
[graphics.DrawLine(Pens.Blue, 10, 10, 20, 10);
```

tekent een lijn met een blauwe pen van punt (10,10) naar punt (20,10)

In de volgende opdracht gaan we er stap voor stap mee oefenen.

3.2 Opdracht

Stap 1: Mijn eerste tekening

We gaan in deze eerste opdracht een programma maken dat een aantal figuren op een *Form* tekent wanneer er op een knop gedrukt wordt.

Doe: Maak een nieuw project in Visual Studio en noem dit project ‘MijnEersteTekening’. Hernoem de ‘Form1’ klasse naar ‘TekeningForm’. Zet een knop onderaan het Form, noem deze knop ‘draw-Button’ en zet er een label met de tekst ‘Draw’ op.

Laat het programma de volgende code uitvoeren als er op de knop gedrukt wordt (lees de uitleg in het commentaar):

```
// Vraag het Graphics object op, dat bij dit form hoort.  
// Met dit graphics object kunnen we op het form tekenen.  
Graphics graphics = CreateGraphics();  
// Na het opvragen van het Graphics object kunnen we gaan tekenen.  
  
int breedte = 100;  
int hoogte = 50;
```

```
// Teken een rechthoek op coordinaat (10, 10)
// en een gevulde rechthoek op coordinaat (10, 70).
graphics.DrawRectangle(Pens.Black, 10, 10, breedte, hoogte);
graphics.FillRectangle(Brushes.Blue, 10, 70, breedte, hoogte);
```

Je hebt nu je eerste programma gemaakt dat zelf tekent.

Andere methoden van het *Graphics* object:

- DrawLine(...)
- DrawBezier(...)
- DrawEllipse(...)
- DrawPolygon(...)
- DrawRectangle(...)

Doe: Voer het programma uit en kijk of het doet wat je verwacht!

Alles lijkt goed te gaan, maar... Wat gebeurt er als je na het zien van de tekening:

- het form minimaliseert en weer maximaliseert?
- het form van grootte verandert door de hoek rechts onder te verslepen (eerst heel klein en dan groter maken).
- een window van een andere applicatie over de tekening sleept.

Zoals je zult merken wordt door bovenstaande acties de tekening deels of geheel gewist. Dit is meestal niet de bedoeling. Eens getekend, altijd getekend zou je verwachten.

Om de afbeelding te herstellen kun je natuurlijk opnieuw op de draw knop klikken. Deze zal de tekening opnieuw tekenen (probeer!). Maar... is dit niet een beetje omslachtig? Lees verder om te ontdekken hoe dit beter kan.

Stap 2: Tekenen met het Paint event

In de vorige opgave hebben we gezien dat de tekening beschadigd raakt bij bepaalde acties en dat we de tekening kunnen herstellen door hem opnieuw te tekenen.

De vraag die overblijft is: wanneer moeten we de tekening opnieuw tekenen (lees: herstellen)?

Hiervoor gebruiken we het *Paint* event van het *form*. Ieder *form* bevat een *Paint* event, dat automatisch afgevuurd wordt als het *form* (en de tekening er op) beschadigd raakt. Door aan het *Paint* event een event-handler te hangen kunnen we de tekening herstellen.

Help! Event en event-handler?! Wat was dat ook alweer? Denk even aan de deurbel. Iemand drukt op de bel, en de bel gaat rinkelen (event). Jij hoort de bel rinkelen (event-handler) en gaat de deur openen (invulling event-handler).

In deze opgave gaan we het programma van Stap 1 opnieuw maken, maar deze keer gaan we het *Paint* event gebruiken om te tekenen.

Doe: Maak een nieuw project in *Visual studio* en noem dit project *TekenenMetHetPaintEvent*.

Hernoem de *Form1* class naar *TekeningForm*. Zet een knop onderaan het *Form*, noem deze knop *drawButton* en zet er het label *Draw* op.

Maak vervolgens een event-handler aan voor het *Paint* event van het *TekeningForm*.

Dit kun je doen door in het properties venster van het *TekeningForm* te dubbelklikken op het *Paint* event. Let op dat het bliksempje ingedrukt is, zodat je de events ziet (figuur 3-1).

Visual Studio voegt nu de volgende code toe aan het *TekeningForm*. Dit is de event-handler voor het *Paint* event.

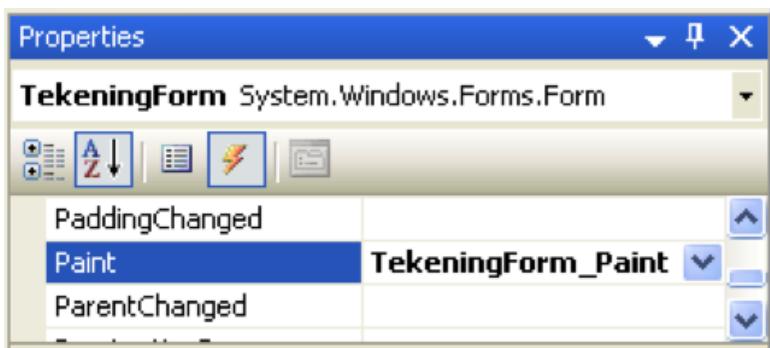


Figure 3-1 Paint Event

```
[private void TekeningForm_Paint(object sender, PaintEventArgs e)
{
}
```

Komt dit je bekend voor? Juist, het event mechanisme is niet nieuw voor je. Je hebt al events gebruikt bij het koppelen van acties aan het indrukken van een knop.

Verder met tekenen. Zet de code voor het tekenen (zie Stap 1) in de event-handler van het *Paint* event. Je krijgt dan

```
private void TekeningForm_Paint(object sender, PaintEventArgs e)
{
    // Vraag het Graphics object op, dat bij dit form hoort.
    // Met dit graphics object kunnen we op het form tekenen.
    Graphics graphics = CreateGraphics();

    // Na het opvragen van het Graphics object kunnen we
    // gaan tekenen.
    int breedte = 100;
    int hoogte = 50;

    // Teken een rechthoek op coordinaat (10, 10)
    // en een gevulde rechthoek op coordinaat (10, 70).
    graphics.DrawRectangle(Pens.Black, 10, 10, breedte, hoogte);
    graphics.FillRectangle(Brushes.Blue, 10, 70, breedte, hoogte);
}
```

Doe: Voer het programma uit en kijk of het doet wat je verwacht!

Blijft de tekening nu wel staan als je de form minimaliseert en maximaliseert of wanneer je de grootte veranderd?

Wat het programma in ieder geval NIET doet, is zich gedragen als het programma uit Stap 1; je tekening wordt direct na het starten op je scherm gezet en de 'Draw' knop heeft geen enkele functie. Dit gaan we anders doen, want het programma moet werken als in opgave 1 gevraagd wordt. Maar voordat we daar aan beginnen, eerst nog iets belangrijks...

Stap 3: Gebruik maken van het juiste Graphics object in de Paint event-handler

Doen we dat dan nog niet? Het programma uit opdracht 2 tekent toch netjes? Ja, maar een werkend programma is niet altijd een goed programma. Wat is er aan de hand?

Kijk in de code van Stap 2. In de code van de *Paint* event-handler staat:

```
[ Graphics graphics = CreateGraphics();
```

Hier staat dus: Vraag het Graphics object op dat bij het TekeningForm hoort. Dit lijkt te kloppen want die willen we opnieuw tekenen.

Het opvragen van het Graphics object kan echter op een betere manier.

De *Paint* event-handler krijgt namelijk informatie mee over het object dat het *Paint* event afgevuurd heeft en dat dus opnieuw getekend moet worden. Deze informatie zit in de parameter van het type *PaintEventArgs* en bevat o.a. het Graphics object dat nodig is om te tekenen. Deze kun je opvragen met:

```
[ Graphics graphics = e.Graphics;
```

Je krijgt dan

```
private void TekeningForm_Paint(object sender, PaintEventArgs e)
{
    // Vraag het Graphics object op van de control, die dit Paint
    // event heeft verzonden. Met het graphics object kunnen we
    // tekenen op dit control.
    // Definitie 'control': Een control is een User Interface object.
    // Voorbeelden van controls: TextBox, PictureBox, Button, Label,
    // Form, Panel, etc
    Graphics graphics = e.Graphics;
    // Na het opvragen van het Graphics object kunnen we
    // gaan tekenen.
    int breedte = 100;
    int hoogte = 50;
    // Teken een rechthoek op coordinaat (10, 10)
    // en een gevulde rechthoek op coordinaat (10, 70).
    graphics.DrawRectangle(Pens.Black, 10, 10, breedte, hoogte);
    graphics.FillRectangle(Brushes.Blue, 10, 70, breedte, hoogte);
}
```

Doe: Vervang de *Paint* event-handler uit je programma van opdracht 2 door de bovenstaande en probeer je programma uit. *Problemen met het programma?*

We hebben nu een programma dat op de goede manier tekent. Maar het programma werkt nog niet als het programma uit Stap 1: de 'Draw' knop heeft nog geen functie. Dat gaan we aanpakken in de volgende opdracht.

Stap 4: De 'Draw' knop gebruiken om te tekenen

We gaan bij deze stap verder bouwen aan de code die gegeven is in 'voorbeeldcodeTekenenMetHetPaintEvent'. We gaan code toevoegen die ervoor zorgt dat er pas getekend wordt nadat er op de 'Draw' knop gedrukt is.

Doe: Maak een kopie van de map 'voorbeeldcodeTekenenMetHetPaintEvent'. Met deze kopie ga je verder werken. Open de gekopieerde solution.

In Stap 2 is gebleken dat we alleen mogen tekenen in de event-handler van het *Paint* event. Teken in de event-handler van de 'Draw' knop is 'verboden'.

Hoe kunnen we er nu toch voor zorgen dat er pas na het drukken op de *Draw* knop getekend wordt en niet direct na de start van het programma?

Een eerste plan:

We introduceren een nieuw bool dataveld *laatTekeningZien* in onze *TekeningForm* class. Dit nieuwe dataveld gaat bijhouden of er getekend mag worden. Initieel mag er niet getekend worden, dus de initiële waarde van *laatTekeningZien* moet *false* zijn. In de *Paint* event-handler gaan we de conditie van *laatTekeningZien* gebruiken voor het wel/niet tekenen van de figuren. Het drukken op de *Draw* knop moet ervoor zorgen dat *laatTekeningZien true* wordt en er dus getekend mag worden.

Als we dit plan uitvoeren krijgen we de volgende code in de klasse *TekeningForm*:

3.2 Opdracht

```
private bool laatTekeningZien; // laat tekening alleen zien als true.

public TekeningForm() {
    InitializeComponent();      // Zorgen voor de juiste initialisatie:
    laatTekeningZien = false; // Initieel geen tekening op het Form.
}

private void TekeningForm_Paint(object sender, PaintEventArgs e) {
    if (laatTekeningZien) {
        Graphics graphics = e.Graphics;
        int breedte = 100;
        int hoogte = 50;
        graphics.DrawRectangle(Pens.Black, 10, 10, breedte, hoogte);
        graphics.FillRectangle(Brushes.Blue, 10, 70, breedte, hoogte);
    }
}

private void drawButton_Click(object sender, EventArgs e) {
    laatTekeningZien = true; // Ervoor zorgen dat er getekend kan worden.
}
```

Doe: Zorg ervoor dat het programma in je gekopieerde project werkt als de bovenstaande code. Wat verwacht je dat het programma doet? Voer het programma uit en kijk of je verwachtingen kloppen. Wat gebeurt er als je de form minimaliseert en maximaliseert na het drukken op de *Draw* knop? Wat denk je dat hier aan de hand is?

Het programma lijkt nu compleet. Toch gaat er nog iets mis waardoor er niet direct getekend wordt na het drukken op de *Draw* knop. Na het minimaliseren en maximaliseren van de form is de tekening er ineens wel. Rara hoe kan dat?

Verklaring: Het minimaliseren en maximaliseren van het *TekeningForm* veroorzaakt automatisch een *Paint* event. Door het *Paint* event wordt de *TekeningForm_Paint* event-handler aangeroepen en die gaan tekenen (we hebben immers zelf *TekeningForm_Paint* aan het *Paint* event verbonden, zie Stap 2).

Als we dus willen tekenen na het drukken op de *Draw* knop moeten we niet alleen *laatTekeningZien* op *true* zetten. We moeten er ook voor zorgen dat er een *Paint* event afgevuurd wordt. Hoe doen we dit?

Je kunt een *Paint* event forceren door de methode *Refresh()* aan te roepen.

Laten we het dit eens toepassen in de *Click* event-handler van de *Draw* knop:

```
private void drawButton_Click(object sender, EventArgs e) {
    laatTekeningZien = true; // Ervoor zorgen dat er getekend kan
                           // worden.

    // Het aanroepen van Refresh() zorgt ervoor ervoor dat het Form
    // als 'beschadigd' wordt gemarkerd. Hierdoor wordt zijn paint
    // event automatisch afgevuurd.
    // Gebruik van de Refresh() methode forceert dus het opnieuw
    // tekenen van het form dmw het Paint event.
    Refresh();
}
```

Doe: Probeer bovenstaande code in je programma uit. Je hebt nu een werkend programma.

Napraten: Slimmeriken hebben misschien het volgende geprobeerd:

```
// VOORBEELD VAN HOE HET NIET(!) MOET...
private void drawButton_Click(object sender, EventArgs e) {
    laatTekeningZien = true; // Ervoor zorgen dat er getekend kan
                           // worden.

    TekeningForm_Paint(wat moet hier??, wat moet hier??);
}
```

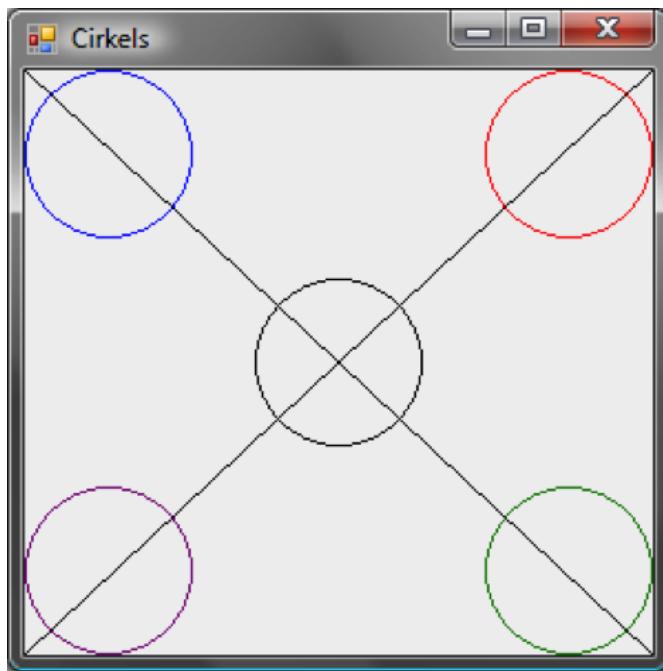


Figure 3-2 Teken App

```
// DIT WAS DUS EEN VOORBEELD VAN HOE HET NIET(!) MOET...
```

Hier kunnen we kort over zijn: het direct aanroepen van event-handlers is niet de bedoeling.

Stap 5: Onderzoek coördinaten stelsel

In de voorgaande opdrachten heb je geleerd hoe je kunt tekenen met een `Graphics` object. In deze opdracht ga je zelf onderzoeken hoe het coördinatenstelsel werkt waarmee getekend wordt. De uitkomst van dit onderzoekje heb je nodig als voorbereiding op de ‘BallenWereld’ opdracht.

Doe: Maak een applicatie die de volgende figuren tekent op een form:

Eisen:

- Elke cirkel heeft als diameter 75
- In elke hoek staat één cirkel.
- In het midden staat een cirkel
- De twee diagonale lijnen kruisen elkaar in het midden.
- Het programma voldoet aan bovenstaande eisen voor elke `Size` van 300x300 en groter.
- Test het programma met `Size “300;300”` en `“500;400”` voor de form.

Let op dat je de `Size` property van je form instelt vóór het runnen van je programma. Je hoeft niet te anticiperen op het resizen van je form tijdens de uitvoer van je programma.

Hints:

- Waar ligt het coordinaat (0,0) in het tekenvlak?
- Wat is het verschil tussen `Width` en `ClientRectangle.Width` en `Height` en `ClientRectangle.Height`?
- De methode voor het tekenen van een cirkel heet niet `DrawCircle(...)`!

Extra info: Gebruik maken van pennen en penselen De voorbeeldprogramma's die je bij de opdrachten hebt gekregen maken gebruik van standaard *pennen* en *penselen*. Deze worden beschikbaar gemaakt door de *Pens* class en de *Brushes* class. Je kunt echter ook zelf pennen en penselen maken. Zo kun je bijvoorbeeld een pen maken met een bepaalde dikte, of een penseel die een plaatje als patroon gebruikt. Dit ligt echter buiten de scope van de lesstof.

Hierbij een voorbeeld waarbij een standaard pen wordt gebruikt.

```
[ graphics.DrawRectangle(Pens.Black, 10, 10, breedte, hoogte);
```

En hetzelfde voorbeeld, maar dan met een zelf gedefinieerde pen.

```
[ int penDikte = 1;
  Pen blackPen = new Pen(Color.Black, penDikte);
  graphics.DrawRectangle(blackPen, 10, 10, breedte, hoogte);
```

3.3 Uitbreiding 1 (niveau 3 / 5)

Maak het formulier iets groter, zodat er onderaan een label past in een gebied waar niet wordt getekend. Voeg onderaan een label toe en schrijf hier de x en y coördinaat naartoe indien ergens op het formulier wordt geklikt. Bijv. x: 20, y:40 Tip: Gebruik hiervoor het *MouseDown* event. Deze handler heeft een parameter genaamd *e* van type *MouseEventArgs*. De X en Y waarde van de coördinaat vraag je op met respectievelijk *e.X* en *e.Y*.

3.4 Uitbreiding 2 (niveau 4 / 5)

Breed het programma uit en teken een rondje waar geklikt wordt met de muis. Onthoud de punten in een lijst. Verbind een punt steeds met het vorige door middel van een lijn. Kies voor de lijn en voor het rondje een willekeurige kleur uit vijf zelfgekozen, in de code vastgelegde, kleuren. Voeg een knop toe, naast het bovengenoemde label, waarmee de toegevoegde punten en de verbindinglijnen weer gewist kunnen worden.

CHAPTER 4

Welke classes zal ik maken?

Programmeren houdt in dat je bedenkt hoe je *de werkelijkheid of de wereld* waar je programma over gaat (deels) gaat *modelleren*. Welke begrippen zijn er nodig binnen het programma (vaak zijn dit de aan te maken *classes*) en welk gedrag verwacht (typisch de te programmeren *methodes*)?

In het eerder beschreven programma over de hero die tegen monsters gaat vechten hebben we een class *Monster* aangemaakt waarin het gedrag van elk monster beschreven werd, en een class *Hero* met daarin methodes die het gedrag van de ene hero (maar het hadden er ook meerdere kunnen zijn) beschrijven.

Als je een programma schrijft voor een *hypoekadviseur* zullen er waarschijnlijk *classes* zijn als *Hypoek* en *Leningdeel* met velden als *rentePercentage*, *looptijd* en *bedrag* en er zullen methodes zijn als *BerekenRenteOverXJaar* (met als parameter over hoeveel jaar het gaat) enzovoort.

De kunst is bij het beginnen van een nieuw stuk software dit soort begrippen boven tafel te krijgen. De software ontwikkelaar moet zich bijna altijd verdiepen in de *materie* waar het programma over gaat (een Game of een Hypoek of wat dan ook).

Bij het lezen van een beschrijving wat het programma moet gaan doen wordt meestal van de *zelfstandige* naamwoorden (*hypoek*, *monster*, *held*) bekijken welke daarvan een class gaan worden en de werkwoorden (*bereken*, *attack*) zijn potentiële methodes. De programmeur of software engineer zal vaak een schema (vaak een diagram met daarin de *classes*) maken om te bepalen hoe het programma er uit gaat zien.

Properties

Properties gebruik je om toegang te verlenen tot afgeschermd (private) fields. Stel, je hebt een Stopwatch klasse, dan zou deze een private field seconds kunnen hebben. Wil je dat dit field alleen gelezen kan worden door andere code, kun je een property hiervoor aanmaken. De conventie is dat fields geschreven worden met kleine letters; properties worden begonnen met een hoofdletter. Zie onderstaand voorbeeld:

```
class Stopwatch
{
    private int seconds;           // Field
    public int Seconds;           // Property
    {
        get { return seconds; }    // Getter
    }
}

Stopwatch sw = new Stopwatch();
int tijd1 = sw.seconds;          // Mag niet, omdat field seconds private is.
int tijd2 = sw.Seconds;          // Mag wel (hoofdletter) omdat de property
                                // public is.
sw.Seconds = 10;                // Mag niet (geen setter)
```

Een andere mogelijkheid is om een field op een bepaalde manier in te stellen. We zouden bijvoorbeeld de stopwatch instelbaar kunnen maken met minuten:

```
class Stopwatch
{
    private int seconds;           // Field
    public int Seconds;           // Property
    {
        get { return seconds; }    // Getter
    }
    public int Minutes;           // Property
    {
        get { return seconds / 60; } // Getter
        set { seconds = value * 60; } // Setter
    }
}

Stopwatch sw = new Stopwatch();
sw.Minutes = 5;                  // Instellen in minuten
int tijd = sw.Seconds;           // Uitlezen in seconden (300)
```

Externe bronnen

CSharp.Net tutorials¹

CodeProject²

MSDN³

Wil je meer lezen over properties dan vind je op MSDN een goede uitleg (negeer voor nu het uitgebreidere voorbeeld onder de kop Example). MSDN over properties⁴

¹<http://csharp.net-tutorials.com/classes/properties/>

²<https://www.codeproject.com/Articles/1006217/Diving-into-OOP-Day-Properties-in-Csharp-A-Practic>

³[https://msdn.microsoft.com/en-us/library/windows/desktop/aa370889\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa370889(v=vs.85).aspx)

⁴<http://msdn.microsoft.com/en-us/library/w86s7x04.aspx>

CHAPTER

6

Private

6.1 Private

- *Information hiding:* Fields binnen een class maken we altijd private.
- Voordeel: beter onderhoudbare software, programmeurs kunnen vanuit andere classes niet zomaar in de *internals* van jouw class, maar gebruiken de daarvoor bedoelde methods en/of properties.

6.2 Methods

Methods die van buitenaf aanroepbaar moeten zijn zijn dus typisch public. Als het methodes betreft die alleen bedoeld zijn voor gebruik binnen de class kun je ze beter private maken.

6.3 Properties

Een property definieert vaak een manier om de waarde van een veld op te vragen of zelfs te veranderen. Het opvragen kan vaak public zijn, maar het is bij de meeste velden niet de bedoeling dat de waarde van buitenaf veranderd kan worden.

```
class Persoon
{
    // Maak Velden private
    private string naam;
    private int Leeftijd;

    public Persoon(string Naam, int Leeftijd)
    {
        this.naam = Naam;
        this.Leeftijd = Leeftijd;
    }
}
```

Het is nu niet mogelijk van buitenaf de waarde van een veld als *Leeftijd* te veranderen:

```
Persoon persoon = new Persoon("Pietje Puk");
int Leeftijd = persoon.Leeftijd; // dit werkt dus NIET!
```

Uiteraard kan er in class *Persoon* een methode gemaakt worden die de waarde van veld *Leeftijd* teruggeeft, een zogenaamde *get-method*:

```
[ public int GetLeeftijd()
  {
    return this.leeftijd;
  }
```

zodat de leeftijd opgevraagd kan worden:

```
[ int leeftijd = persoon.GetLeeftijd();
```

Merk op dat de *leeftijd* van buitenaf dus niet veranderd kan worden, maar alleen opgevraagd!

6.4 Waarom?

Wat is hier nu het voordeel van? Nou, binnen een jaar zal de ontwikkelaar van deze class de eerste klachten krijgen dat niet de leeftijden niet goed berekend worden, aangezien dat de *leeftijd* geen vaste waarde is: op het moment dat de *persoon* jarig is moet de waarde opgehoogd worden. In dit geval kun je zien dat het verstandiger is de *geboortedatum* van de persoon op te slaan (die verandert namelijk niet) en dan wordt bij het opvragen van de *leeftijd* de goede waarde berekend. De programmeur kan nu zonder problemen de class veranderen:

```
[ class Persoon
{
  // velden
  private string naam;
  private DateTime geboortedatum;

  public int GetLeeftijd()
  {
    int leeftijd;
    ... // voeg hier code toe om de leeftijd te berekenen mbv de geboortedatum.
    return leeftijd;
}
```

Doordat het veld *leeftijd* *private* was kan de programmeur dit aanpassen zonder dat er elders problemen ontstaan in code die hier gebruik van maakt, externe code roept namelijk de methode *GetLeeftijd()* aan en die zal na de wijziging zonder problemen werken.

6.5 Encapsulation

Stel dat er een bug zit in (de waarde van velden van) een bepaalde class, dan is het ook prettig te weten dat (in geval van *private* velden) de bug ergens in de class moet zitten. Dit wordt Encapsulaton genoemd: een stukje gedrag van een programma wordt afgeschermd. Hierdoor wordt het makkelijker een deel van je programma te hergebruiken.

Encapsulatie betekent kortweg dat een groep fields, methodes en overige eigenschappen gezien worden als een enkel, afgebakende eenheid of object. Dit klinkt wat droog, dus een andere bewoording die mogelijk duidelijker is door encapsulatie te zien als het vermogen van een klasse om fields en methodes die niet interessant zijn voor anderen, te verbergen.

De beste reden om bepaalde onderdelen van je klasse af te schermen is dat je code makkelijker in het gebruik wordt. Klassen gebruiken *private* fields om hun toestand bij te houden; hoeveel levens heb ik nog, hoe snel mag ik bewegen, et cetera. Deze informatie is voor andere code niet per sé interessant, maar wel essentieel voor de werking van de klasse. Als iedereen zomaar die fields aan zou kunnen passen, wordt de werking van je klasse een stuk onbetrouwbaarder en willekeuriger: wanneer je zelf de enige bent die het aantal levens aan kunt passen, weet je ook precies waar in je code het voor kan komen dat je levens op 0 worden gezet. Dit voordeel valt weg als iedereen het aantal levens aan kan passen.

6.6 External References

Over private¹

Techopedia²

¹<https://softwareengineering.stackexchange.com/questions/143736/why-do-we-need-private-variables>

²<http://www.techopedia.com/definition/3787/encapsulation-c>

Scheiding tussen GUI en Domain

In dit hoofdstuk bespreken we een aantal principes bij de scheiding tussen GUI-klassen en domeinklassen (domain classes). Alvorens deze principes te presenteren leggen we eerst uit wat een domeinklasse is. We eindigen dit hoofdstuk met illustraties waar de beoogde scheiding goed of slecht (lees: minder goed) is uitgevoerd.

7.1 Domein

Elke applicatie (of app) is een hulpmiddel om zekere processen sneller, prettiger of anderszins beter te laten verlopen. Deze processen hebben betrekking op het creëren, inspecteren, wijzigen of verwijderen van informatie. Bijvoorbeeld het aanvragen van een nieuwe bankrekening, het opvragen van het saldo van een bankrekening, het overmaken van geld van de ene naar de andere bankrekening, of het afscheid nemen als klant van een bank. In deze gevallen gaat het steeds om informatie uit het applicatiedomein van financieel verkeer. De informatie is gerelateerd aan objecten uit dit applicatiedomein: zoals Bankaccount, Bank, Client en Transaction. We noemen dit objecten van domeinklassen. Daar tegenover staan objecten die een middel vormen

- om domeinobjecten te kunnen visualiseren in een GUI, of
- om informatie in objecten te kunnen bewaren in een database of bestand, of
- om informatie in objecten over een netwerk te kunnen versturen of ...

Deze drie laatstgenoemde objecten behoren niet tot het domein.

We hebben voor het domein van financieel verkeer de domeinklassen Bankaccount, Bank, Client, Transaction geïntroduceerd. Voor een spel met helden en monsters zouden de domeinklassen Hero, Monster, Player en Game eenzelfde centrale rol kunnen vervullen. Deze klassen herbergen de belangrijke informatie om zo'n spel in een GUI zichtbaar te kunnen maken.

Samenvattend: Domeinklassen herbergen de relevante informatie uit het beoogde applicatiedomein.

Challenge1: welke domeinklassen zou jij willen introduceren binnen de context van het spel gal-gje? Challenge2: welke domeinklassen zou jij willen introduceren binnen de context van het verzenden, inzien en verwijderen van e-mails?

7.2 Principles

Wij adviseren om je te houden aan onderstaande vier principes, ter wille van een goede scheiding tussen de programmacode in een GUI en de programmacode in de domeinklassen. Deze lijst met principes is niet compleet, maar wordt in de loop van je studie uitgebreid en verder genuanceerd.

1. Gebruik geen GUI-objecten in domeinklassen. Want dan kun je zo niet gemakkelijk overstappen naar een andere GUI (bijv. WindowsForm, Web GUI, smart phone, andere vormgeving van de GUI, verschillen in GUI per soort gebruiker). Neem binnen een domein-object ook geen plaatjes op (later wil jewel eens een ander plaatje gebruiken). Je software wordt daarmee flexibeler en beter onderhoudbaar.
2. Wees voorzichtig met het opnemen van zelf gedefinieerde reken- of validatiemethoden binnen een GUI-klasse (zie 3.)
3. Beperkingsregels (zoals het saldo van een bankrekening mag niet lager worden dan de kredietlimiet) en berekeningen (zoals hoeveel rente krijg je per maand bijgeschreven) moet je voor 100% borgen binnen de betreffende domeinklasse die over alle vereiste informatie beschikt.
4. Als je van domeinobjecten een tekstrepresentatie in de GUI wilt kunnen opnemen, dan is het handig om de ToString-methode van de betreffende domeinklasse te 'overriden'.

7.3 Illustraties

Deze vier principes illustreren we aan de hand van de bankrekening-casus.

ad 1. Fout: Binnen een ChangeBalance-methode van de Bankaccount-klasse een messagebox (MessageBox.Show) activeren zodra er wordt geprobeerd om teveel geld van het BankAccount-object af te schrijven.

```
public class Bankaccount {
    private decimal balance;
    private decimal threshold; // not negative
    ...
    public void ChangeBalance(decimal amount) {
        if (balance + amount < -threshold) {
            MessageBox.Show("withdrawal of " + amount + " is not allowed");
        } else {
            balance -= amount;
        }
    }
}
```

Goed: De ChangeBalance-methode van de Bankaccount-klasse retourneert bijvoorbeeld een string. De betreffende string represeneert of het opnemen van het geldbedrag al dan niet is geaccepteerd. Vervolgens kan deze returnwaarde naderhand in de GUI binnen een MessageBox als een message worden getoond.

```
public class Bankaccount {
    private decimal balance;
    private decimal threshold;
    ...
    public string ChangeBalance(decimal amount) {
        if (balance + amount < -threshold) {
            return "withdrawal of " + amount + " is not allowed";
        } else {
            balance -= amount;
            return "withdrawal is succeeded";
        }
    }
}
```

Merk op, in plaats van de string als returnwaarde kan er ook met een simpele bool worden gewerkt.

ad 2. Fout: Binnen de GUI-klasse wordt een hulpmethode CheckValidTransaction gedefinieerd waarbinnen alvast wordt gecontroleerd of het overmaken van geld, van de ene bankrekening naar een andere bankrekening uitvoerbaar is.

```
// een methode binnen de GUI-klasse
bool CheckValidTransaction(Bankaccount ba, decimal amount) {
    if (ba.getBalance() + amount < -ba.getThreshold()) {
        return false;
    } else {
        return true;
    }
}
```

Goed: Stuur het verzoek tot overmaken door naar het betreffende Bank-object van de bankrekening waar het geld van wordt ageschreven. Later rapporteert het Bank-object of het overmaken is geslaagd, bijvoorbeeld door middel van een string (zie ook ad 1.)

```
[[[
public class Bank {
    ...
    public string Transfer(string from, string to, decimal amount) {
        // check if bankaccount number from exists
        Bankaccount baFrom = GetBankaccount(from);
        if (baFrom==null) {
            return "Bankaccount " + from + " does not exist.";
        } else {
            // check if bankaccount number to exists
            Bankaccount baTo = GetBankaccount(to);
            if (baTo==null) {
                return "Bankaccount " + to + " does not exist.";
            } else {
                string result = baFrom.ChangeBalance(-amount);
                if (result == "withdrawal is succeeded") {
                    baTo.ChangeBalance(amount)
                }
                return result;
            }
        }
    }
}]
```

ad 3.

Neem even aan dat er bij een bank geen twee klanten mogen voorkomen met dezelfde combinatie van naam/adres/geboortedatum.

Fout: Binnen de Bank-klasse wordt er, zodra er een nieuwe klant wordt toegevoegd, niet gecontroleerd of er al een klant bestaat met dezelfde naam, adres en geboortedatum.

```
[[[
public class Bank {
    private List<Client> clients;
    ...
    // andere gegevens van de klant zijn nu even genegeerd:
    public void AddClient(string name, string place, DateTime birthdate) {
        Client client = new Client(name, place, birthdate);
        clients.Add(client);
    }
}]
```

Goed: Binnen een AddClient-methode van de Bank-klasse wordt eerst gecontroleerd of er al een

klant bestaat met dezelfde naam, adres en geboortedatum. Zo ja, dan wordt het nieuwe Client-object niet gecreëerd en geregistreerd; anders wel. De AddClient-methode heeft als returnwaarde het nieuwe Client-object. Als de returnwaarde null is, is er geen Client-object gecreëerd.

```
public class Bank {
    private List<Client> clients;
    ...
    // andere gegevens van de klant zijn nu even genegeerd:
    public Client AddClient(string name, string place, DateTime birthdate) {
        Client client = GetClient(name, place, birthdate);
        if (client != null) return null;
        client = new Client(name, place, birthdate);
        clients.Add(client);
        return client;
    }
}
```

ad 4. Binnen de GUI wordt een lijst met gegevens van alle bankrekeningen van een specifieke klant getoond. Dan is het handig als de ToString-methode van de Bankaccount-klasse wordt geverifieerd; bijvoorbeeld door te volstaan met het rekeningnummer en de naam van de eigenaar van de bankrekening:

```
public class Bankaccount {
    private string nr;
    private Client owner;
    ...
    public override string ToString() {
        return this.nr + " : " + this.owner.GetName();
    }
}
```

Deze ToString-methode kan vervolgens binnen de GUI worden aangeroepen.

8

CHAPTER

Text File Handling

8.1 Terminologie

- Wat is de betekenis van de volgende woorden?
 - File, Bestand
 - Directory, Pad, Folder, Map
 - UNC
 - URL
- Hoe ziet een pad er uit in Windows Verkenner, Command Prompt, Linux, Mac OSX?

8.2 File handling klassen

Het .NET framework biedt een aantal klassen* om met bestanden te kunnen werken:

- File
- Directory
- DirectoryInfo
- Path

8.3 Andere bronnen

MSDN¹

¹<https://msdn.microsoft.com/en-us/library/2kzb96fk.aspx>

Exception Handling

De sleutel tot succes ligt in alle momenten waarop je hebt gefaald. (Jean Marie Molina)

FileNotFoundException, NullPointerException, enzovoort. Al die fouten die jij ziet in de Visual Studio debugger ziet de gebruiker ook. Maar als de gebruiker ze ziet dan crasht zijn programma! Als software engineer dien je mogelijke exceptionele situaties te voorkomen. C# heeft hiervoor de keywords try, catch en finally.

Regelmatig wil je de structuur van je code niet helemaal aanpassen aan exceptionele situaties die je (inderdaad) als een *uitzondering* wilt beschouwen. Een voorbeeld is wanneer je programma een connectie met een database nodig heeft, of requests naar internet stuurt: je wil dan niet elke keer checken of de connectie naar database of internet nog wel werkt, maar **als** de connectie verloren is gegaan dan moet je programma daar wel mee om kunnen gaan. Je kunt dit doen door om je code een try-catch-clause te zetten. Het idee ziet er als volgt uit:

```
... // code die verbinding maakt
try {
    // code die er van uit gaat dat de connectie werkt:
    ...
    // als er tijdens het uitvoeren van deze code
    // een exception optreedt springt C# meteen
    // naar het vangnet hieronder dat bijvoorbeeld
    // de gebruiker meldt dat de verbinding
    // verbroken is en dat ie het bijvoorbeeld
    // nog eens moet proberen.

} catch(Exception exc)
{
    ...
    // code die uitgevoerd wordt wanneer er
    // een probleem optreedt!
}
```

Andere bronnen

Exceptions at CSharp-station¹

Exceptions at MSDN²

¹<http://www.csharp-station.com/Tutorial/CSharp/Lesson15>

²<https://msdn.microsoft.com/en-us/library/ms173162.aspx>

CHAPTER 10

Enum

10.1 Definie van Enum

- Enumeraties of kortweg enum's stellen je in staat items op een gestructureerde, geordende manier voor te stellen.
- Een enumeratie zorgt ervoor dat de elementen aan te spreken zijn met een naam, maar worden intern genummerd (standaard vanaf 0).
- Met een enumeratie heb je onmiddellijk de Visual Studio Intellisense ter beschikking en behoed je jezelf voor tikfouten en logische fouten.

10.2 Voorbeelden

```
enum Dag
{
    Zondag,
    Maandag,
    Dinsdag,
    Woensdag,
    Donderdag,
    Vrijdag,
    Zaterdag
}
```

Dan is mogelijk:

```
Dag d;
d = Dag.Woensdag;
```


CHAPTER 11

Override ToString()

```
[class Persoon {  
    // Field  
    private string naam;  
  
    // Property  
    public string Naam  
    {  
        get { return this.naam; }  
    }  
  
    // ctor  
    public Persoon(string naam)  
    {  
        this.naam = naam;  
    }  
  
    public override string ToString()  
    {  
        return this.Naam;  
    }  
}
```

Tip bij het programmeren: zet je eigen objecten in de user interface. Hiermee wordt bedoeld dat je objecten zelf in de UI zet, geen strings of andere variabelen. Bijvoorbeeld om een Persoon-object aan een listbox toe te voegen:

```
[listBox1.Items.Add(new Persoon("Sjakie"));
```

Gebruik casting om het object uit een UI-control te halen:

```
[Persoon p = (Persoon)listBox1.Items[2];
```

Programmeer een `ToString()`-methode in al je classes om te zorgen dat je altijd een goede tekstuele representatie van je objecten hebt.

video 57 van kudvenkat over `override ToString()`¹

¹<https://www.youtube.com/watch?v=MwPZLPNR3ns&t=0s&list=PLAC325451207E3105&index=57>

CHAPTER 12

Class Diagram

Al eerder heb je class diagrams gezien: plaatjes waarin classes als rechthoeken worden getoond.

In figuur 12-1 zie je steeds twee classes, A en B. De relatie tussen A en B is steeds anders: De pijl geeft een relatie aan tussen de classes, inclusief richting (class A kent class B).

We gaan nu kijken hoe die relaties naar C#-code vertaald kunnen worden:

Mogelijk bij (a) behorende C#-code

```
public class A {  
    // Fields
```

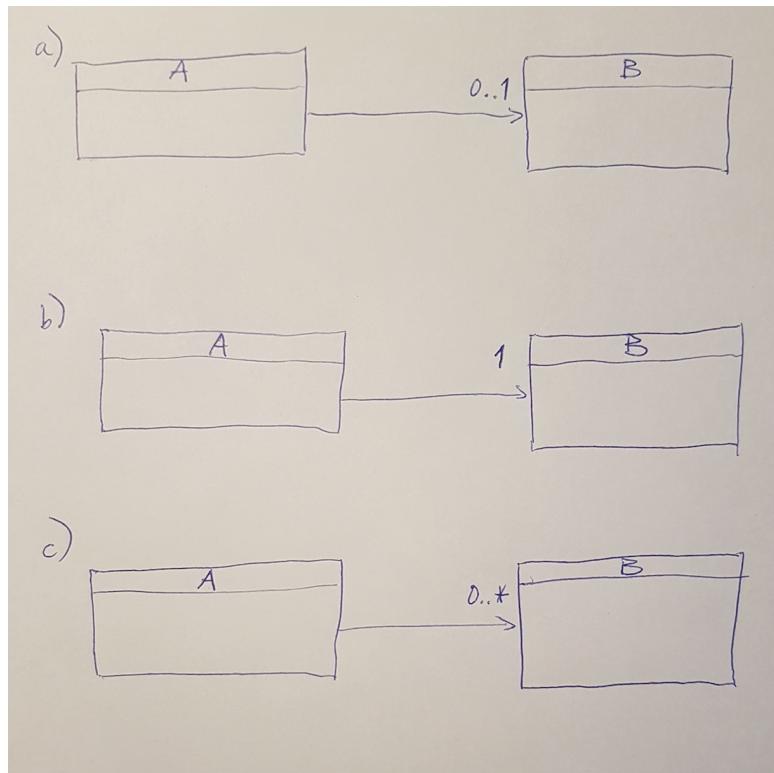


Figure 12-1 Relation in class diagram

```

    private B b;
}

```

Hierbij *kent* class A *class* B. Het Field krijgt vaak de naam van de class maar dan beginnend met kleine letter. De waarde van b kan null zijn of een object van type B.

In situatie (b)

```

public class A {
    // Fields
    private B b = new B();
}

```

Hierbij *kent* class A *class* B. De waarde van b wordt direct ingevuld, deze zal dus niet null zijn.

Bij (c) behorende C#-code:

```

public class A {
    // Fields
    private List<B> bs = new List<B>();
}

```

Een object van type A kent 0 of meer (vanwege de $0..*$) objecten van type B.

Voor de naam van het Field (hier *bs*) wordt doorgaans het meervoud gekozen van *b*. Stel dus bijvoorbeeld dat class B niet B zou heteren maar *BattleRager*, dan zou het Field *b* in plaats daarvan *battleRager* heteren en het Field *bs* zou *battleRagers* worden.

Merk op dat in plaats van een List ook een Array gebruikt kan worden.

Part III

Challenges

13

CHAPTER

Object Oriented Invul-Oefening

Leerdoelen	Class, Instance, Method, Operation, Object, Attribute, Field.
Vereiste voorkennis	Basiskennis over objecten.
Challenge Type	Invulopdracht, kennis

Vul de volgende woorden op de juiste plaats in: Class, Instance, Method, Operation, Object, Attribute. Let op: een woord kan meerdere malen in de tekst voorkomen, en soms vul je de meer- voudsvorm van deze woorden in.

Een programmeur moet van zijn baas binnen een game de "Karakter" gaan programmeren. "Karakter" is reeds gespecificeerd door middel van een kaartje waar alle verantwoordelijkheden op staan. Ook krijgt hij een class diagram met een duidelijk overzicht van de, waaronder MoveForward en ValDoodNeer) en (zoals bijvoorbeeld AantalLevens, HaarKleur en AantalWapens).

De programmeur neemt het kaartje en opent het bestaande project in Visual Studio. Daar gaat hij dan de nieuwe in programmeren. Als eerste programmeert hij Fields voor AantalLevens en HaarKleur en vervolgens maapt hij de MoveForward en ValDoorNeer naar C#-.....

Als hij de helemaal geprogrammeerd gaat hij deze testen door er met de new-operator een van aan te maken. Hij roept de MoveForward aan om te testen of het karakter de goede kant op beweegt. Ja, het werkt! Hij maakt nog een van "Karakter" aan om te testen of het programma dan nog steeds werkt. Ja! Met een voldaan gevoel gaat de programmeur aan het eind van de dag naar huis.

Challenge BattleSim

Datum	Week 11/12
Versie	1 - Inge van Engeland
Leerdoelen	Class, Property, Constructor, private/public, UI separation.
Vereiste voorkennis	Method, GUI, Basic Types, If.
Challenge Type	Programming

14.1 Inleiding

In het spel BattleSim kun je met twee spelers tegen elkaar vechten. In deze versie is de linkerspeler een Knight en de rechterspeler een Ranger. Beide spelers hebben elk 100 hitpoints.

Het is een turn-based spel, dus om beurten vallen de spelers elkaar aan. De sterkte van de aanval ligt tussen de 0 (een misser) en de 30. Als de speler een klap van meer dan 25 geeft, dan heb je een "Critical hit". Op het moment dat het aantal hitpoints 0 of minder is, dan heeft die speler helaas verloren.



Figure 14-1 battlesim

14.2 Opdracht

Programmeer de BattleSim. Figuur 14-1 is een voorbeeld van hoe het scherm eruit kan zien.

Opdracht BattleSim

- Maak een klasse Speler, deze heeft een naam en een aantal hitpoints.
- Maak de properties en de constructor voor deze klasse.
- De klasse speler heeft een methode om schade te ontvangen en een methode om schade te geven.
- Een aanval geeft 0-30 schade.
- Als de aanval 0 is, is het een misser (geef een bericht dat het een misser was). Gebruik hiervoor een exception.
- Als de aanval 25+ is, dan is het een “Critical Hit” (geef een bericht). Gebruik hiervoor een exception.
- Na elke aanval worden de hitpoints bijgewerkt op het scherm.
- Als het aantal hitpoints 0 of minder is, dan heeft die speler verloren (gebruik een exception)
- De karakters om beurten aanvallen.
- Als de ene speler aan de beurt is, kan de andere speler niet op de “Attack” knop klikken.

14.3 Extra

Verzin functies om je BattleSim uit te breiden. Denk bijvoorbeeld aan:

- Plaatje dat verandert als de speler gewonnen / verloren heeft
- Startscherm waarin je een karakter kunt kiezen en een naam kunt geven
- Een “Explore” knop erbij, waarmee je bijvoorbeeld een health potion kunt vinden.
- Hou de hitpoints bij in een Progress Bar.
- Wapens, armor etc.

Challenge Vier op een rij

Leerdoelen	Class, method, constructor, object, private, enum.
Extra	Algoritme,
Vereiste voorkennis	Kennis over objecten, Method, If.
Challenge Type	Programming.

15.1 Vier op een rij

Maak het spel 4 op een rij waarbij je tegen de computer kunt spelen, maar houdt wel rekening met onderstaande eisen:

1. Programmeer class *Zet* met twee integer properties *Rij* en *Kolom* (het property *Rij* is read-only, deze wordt uitgerekend omdat de stenen naar beneden vallen). De constructor accepteert een referentie naar klasse *Spel* (zie hieronder) en een integer-kolom.

2. Een enum *Veld* met mogelijke waarden Rood en Geel.

```
[enum Veld {Rood,Geel}
```

3. Klasse *Spel* met intern een private array van 6 bij 7 (tweedimensionaal array)

```
[Veld[] bord:  
bord = new Veld[6,7];
```

4. Het is een console-applicatie (dus geen Forms gebruiken). Het speelbord hoeft niet te worden afgedrukt. Mag wel, maar dan met *Console.WriteLine* (loop dan door het array heen en druk het stap voor stap af)

5. De klasse *Spel* heeft methoden als *BedenkEenZet()* om de beste zet voor de computer te bedenken en *AccepteerEenZet(Zet z)* (om een zet van de gebruiker te accepteren).

■ Note

De methode *BedenkEenZet()* kan in eerste instantie op zoek gaan naar de eerste de beste vrije kolom. Indien er geen vrije kolom meer is dan kan de methode *GelijkSpel()* worden aangeroepen die het spel stopt. Maak private hulpmethoden zoals bool *HeeftSpelerGewonnen()* en bool *HeeftComputerGewonnen()* die je zelf aanroept.

```
[private Random Randje;  
  
public Zet BedenkEenZet()  
{
```

```
    return new Zet(Randje.Next(7));  
}
```

16

CHAPTER

Challenge Galgje

Leerdoelen	Constructor, read file, private, class diagram.
Extra	Algoritme,
Vereiste voorkennis	Class, object.
Challenge Type	Programming.

Galgje Maak het spel galgje waarbij je als gebruiker tegen de computer een woord moet raden.
Technische en functionele eisen:

1. Programmeer de klasse Woord die het te raden woord bevat. Programmeer bijvoorbeeld een constructor Woord(string w).
2. Lees het te raden woord in uit een tekstbestand en maak vervolgens in C# een Woord-object aan.
3. Programmeer de klasse Galgje die onder meer een veld van het type Woord heeft.
4. Zorg ervoor dat het form alleen een dataveld van het type Galgje heeft, het form heeft GEEN dataveld van het type Woord.

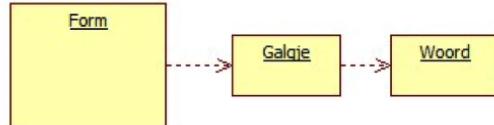


Figure 16-1 galgje

Challenge Ontwerp Webshop Spierballetje

Leerdoelen	Ontwerp classes.
Vereiste voorkennis	Class, object.
Challenge Type	Technisch ontwerp, discussion.

17.1 De opdracht

Webshop spierballetje.com verkoopt spullen voor krachtrainers en bodybuilders. Ze verkopen ondermeer halters (gewichten), kettlebells, sportkleding, schoenen en boeken over voeding en bodybuilding.

Artikelen die ze verkopen zijn onder te verdelen in categorieën kleding, gewichten, boeken en diversen. Elk artikel heeft een inkoopprijs, verkoopprijs en een titel en omschrijving.

Functionaliteiten zijn dat klanten kunnen bestellen (artikelen in winkelwagen kunnen plaatsen), betalen, account aan kunnen maken. Klanten krijgen automatisch mails over de status van de bestellingen. Ze krijgen een factuur via de mail.

Medewerkers van de webshop kunnen met de software (een backend-applicatie) inkopen administreren, btw-aangiften afhandelen en lijsten van leveranciers bijhouden. Ook kunnen ze artikelen op de site updaten.

Bezoekers van de site kunnen reviews plaatsen voor artikelen. Klanten kunnen vragen stellen via de online chatbox in de webshop. Bezoekers van de webshop kunnen artikelen zoeken op naam, prijs, titel enz. Ze kunnen artikelen sorteren op prijs of op meestverkocht.

Klanten kunnen zich inschrijven voor een nieuwsbrief.

Aan jouw de taak om als software engineer vanuit bovenstaande specificatie de te programmeren klassen te ontwerpen. Hint: minimaal zijn er al gauw minimaal 5 tot 20 a 25 klassen te bedenken.

18

CHAPTER

Challenge Platenmaatschappij

Leerdoelen	Class.
Vereiste voorkennis	Basiskennis over objecten.
Challenge Type	Technisch ontwerp.
Extra	Realiseren van het ontwerp.

Platenmaatschappij D'n Gulden Schijf wil een applicatie ontwikkelen om hun contactpersonen in een database bij te houden.

De contactpersonen van de maatschappij bestaan uit artiesten, bands, managers en ook uit leveranciers die bijvoorbeeld kantoorartikelen en grondstoffen voor het bedrijf leveren. Van elke artiest of band moet naast de gebruikelijke naam-en-adres-gegevens ook worden bijgehouden welke platen (songs) ze hebben uitgebracht. Elke song bestaat uit een titel, het jaar van uitgave en een tijdsduur (in minuten + seconden). Als je in het systeem door songs bladert dan kun je elke song aanklikken om af te spelen. Hij speelt dan een hoge kwaliteit-MP3 af van die song.

Zoekfunctie: binnen het systeem moet kunnen worden gezocht op naam van band, manager, leverancier of naam van artiest. Bij de gegevens van de managers moet worden bijgehouden wat hun uurtarief (in euro) is.

Als een contactpersoon een leverancier is dan staat er naast de naam+adresgegevens ook bij wat de gemiddelde levertijd (tijd in dagen) is van de producten van die leverancier. Bij een band moet ook nog worden opgeslagen of er een speciaal instrument bij zit (ja/nee-veld). Speciale instrumenten zijn alle instrumenten die niet gelijk zijn aan drums, keyboard, basgitaar en gitaar. Voorbeelden van speciale instrumenten zijn bijvoorbeeld een dwarsfluit, harp, enz.

Het systeem moet worden beveiligd middels een user+wachtwoord-systeem. Users (gebruikers) kunnen zich aanmelden met een gebruikersnaam en wachtwoord waarna ze toegang hebben tot alle gegevens in het systeem.

D'n Gulden Schijf Accounts Marketing Opportunities Workflow More + Social Users Search for contact, action, deal... [?] 7 [] chames +

All Contacts Advanced Search Clear Filters Columns ?					
	Name	Phone	Last Updated	Lead Source	
<input type="checkbox"/>	Amanda Bailey	(555) 555-5555	Jan 11, 2013 7:23:35 AM		
<input type="checkbox"/>	Lauren Deimel	(555) 555-5555	Jan 11, 2013 7:23:35 AM		
<input type="checkbox"/>	Ryan Coraci	(555) 555-5555	Jan 11, 2013 7:23:35 AM		
<input type="checkbox"/>	Brooke Brewer	(137) 124-2526	Jan 9, 2013 8:29:39 AM	Facebook	
<input type="checkbox"/>	Nina Yates	(850) 195-1205	Jan 9, 2013 8:15:29 AM	None	
<input type="checkbox"/>	Rachel Woodward	(381) 979-8899	Jan 9, 2013 8:04:26 AM	None	
<input type="checkbox"/>	Haley Dudley	(889) 519-5713	Jan 9, 2013 8:02:37 AM	Google	
<input type="checkbox"/>	Bree Matthews	(938) 541-0948	Jan 9, 2013 7:36:46 AM	Google	
<input type="checkbox"/>	Marissa Romero	(555) 555-1111	Jan 7, 2013 10:51:43 AM		
<input type="checkbox"/>	Dylan Carlson	(448) 672-9972	Jan 4, 2013 12:11:47 PM	Facebook	
<input type="checkbox"/>	Rory O'Neill	(205) 832-2034	Jan 4, 2013 11:31:40 AM	None	
<input type="checkbox"/>	Bernard Sims	(881) 443-1481	Jan 4, 2013 7:21:49 AM	Google	
<input type="checkbox"/>	Kaitlin Thomas	(649) 724-9647	Jan 3, 2013 5:59:30 AM	Google	

Active Users [+] Chris Hanes Chloe Greigo [x]
Tag Cloud [?] Just Me | All Users [?]
#research #hardware #met-in-person
#accessories #new #popular
#successful #partner #important
#servers
Message Board [+] Please enter a message of the day!
Chat [+] Quick Contact [+] Note Pad [+] Feel free to enter some notes!

Figure 18-1 platenmaatschappij

Challenge BankRekening

Niveau	3 of 5
Leerdoelen	Class, constructor, object, this, static.
Vereiste voorkennis	Basiskennis over objecten.
Challenge Type	Realiseren.

In deze opdracht maak je een toepassing die een sterk vereenvoudigde bank met maar twee bankrekeningen voorstelt. Hieronder zie je een voorbeeld van het scherm van de toepassing.

Met deze toepassing kan men:

- Geld storten op de rekening links of rechts. In de TextBoxes vul je het bedrag in, vervolgens klik je op de Button “Storten”. Alleen een positief bedrag kan gestort worden. Indien er een negatief saldo zou ontstaan, of als er iets anders ingevuld is dan positieve getallen wordt de transactie niet uitgevoerd en wordt een foutmelding gegeven door middel van een MessageBox.
- Geld opnemen van de rekening links of rechts. In de TextBoxes vul je het bedrag in, vervolgens klik je op de Button “Opnemen”. Indien er een negatief saldo zou ontstaan, of als er iets anders ingevuld is dan positieve getallen wordt de transactie niet uitgevoerd en wordt een foutmelding gegeven door middel van een MessageBox.
- Geld overboeken van links naar rechts (of van rechts naar links). In de TextBoxes links (rechts) vul je het over te maken bedrag in daarna klik je op de Button » («). Indien er een negatief

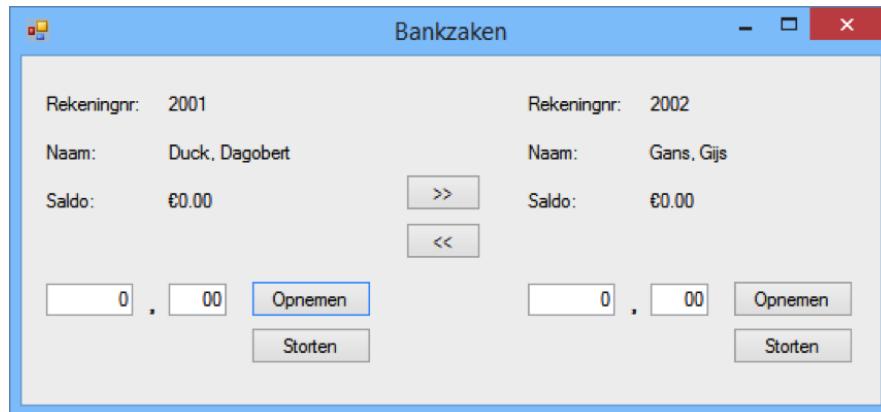


Figure 19-1 bankrekening

saldo zou ontstaan bij de betalende partij, of als er iets anders ingevuld is dan positieve getallen wordt de transactie niet uitgevoerd en wordt een foutmelding gegeven door middel van een MessageBox.

Het weergegeven saldo wordt na iedere transactie natuurlijk netjes aangepast.

19.1 Opdracht

STAP 1: HET FORMULIER

Maak een nieuw Windows Forms project aan dat je bijvoorbeeld Bankzaken noemt. Geef het automatisch aangemaakte formulier Form1 een meer betekenisvolle naam, bijvoorbeeld BankrekeningForm. Pas ook de Property Text van het formulier aan zodat er een betere naam in de titelbalk van de applicatie komt te staan.

STAP 2: DE GUI

Sleep de benodigde componenten op het formulier. Het hoeft niet precies zoals in het voorbeeld staat, maar bedenk wel vooraf hoe je de vereiste functionaliteit koppelt aan de componenten. Geef de componenten een betekenisvolle naam bijvoorbeeld btnStortenLinks en txtEuroRechts.

STAP 3: DE KLASSE BANKREKENING

Voeg een nieuwe klasse toe en noem deze Bankrekening. Elke bankrekening heeft een rekeningnummer, staat op naam van een persoon en heeft een saldo. Zorg er voor dat de klasse Bankrekening-Fields heeft om de benodigde gegevens op te slaan en verder de volgende velden heeft:

```
// Fields
private int rekeningnummer;
private string naam;
private int saldo; // het saldo in hele centen
private static int volgendeVrijeRekeningnummer = 2001;
```

Properties

Programmeer de volgende attributen als READ ONLY (!) property's:

- Rekeningnummer (type int)
- Naam (type string)
- Saldo (type int)

```
// Methods
public void NeemOp(int bedrag)
{
    // bedrag in hele centen, negatieve bedragen worden genegeerd.
    // vul zelf in
}

public void Stort(int bedrag)
{
    // bedrag in hele centen, negatieve bedragen worden genegeerd.
    // vul zelf in
}

public void MaakOver(Bankrekening andereRekening, int bedrag)
{
    // bedrag in hele centen, negatieve bedragen worden genegeerd.
    // vul zelf in
```

```
[ ]}
```

Maak de implementatie van de methodes waarbij “vul zelf in” staat, zelf af.

Static variabele

Merk op dat er static staat voor de variabele volgendeVrijeRekeningNummer en dat deze variabele op 2001 wordt geïnitialiseerd. Dat er static staat betekent dat dit een zogenaamde klassenvariabele is. Een klassenvariabele bestaat al voordat er een instantie gemaakt is van de klasse en wordt bij het opstarten van de applicatie geïnitialiseerd en dus niet pas bij het instantiëren (aanmaken) van het object van die klasse, zoals normale variabelen. Het voordeel van een klassevariabele is dat deze voor alle instanties van deze klasse dezelfde waarde heeft. Dezelfde klassevariabele wordt dus door alle instanties van deze klasse gebruikt.

Elke nieuwe bankrekening moet natuurlijk een uniek rekeningnummer hebben. Bij deze bank zijn dat de nummers vanaf 2001. De eerste bankrekening, die gecreëerd wordt, krijgt rekeningnummer 2001, de volgende rekeningnummer 2002, etc. Het bepalen van het volgende vrije rekeningnummer gebeurt in de Constructor.

Constructors

Een Constructor is een speciale methode met dezelfde naam als de klasse die wordt uitgevoerd als een object van die aangemaakt wordt en dient om de Fields of Properties van de klasse te initialiseren. We hebben twee Constructors nodig. Een om een Bankrekening aan te maken als de naam van de rekeninghouder bekend is en het beginsaldo 0 en een voor het geval dat er wel sprake is van een beginsaldo, bijvoorbeeld als onderdeel van een wervingsactie van de bank.

```
// Constructors
public Bankrekening(string naam)
{
    this.naam = naam;
    saldo = 0;
    rekeningnummer = volgendeVrijeRekeningnummer;

    // we hogen het volgende vrije rekeningnummer met 1 op zodat de
    // volgende bankrekening een nummer krijgt dat 1 hoger is dan
    // deze bankrekening.
    volgendeVrijeRekeningnummer++;
}

public Bankrekening(string naam, int saldo)
{
    // vul zelf in
}
```

Note Het is in veel C# teams gebruikelijk om de Constructors na de Properties en voor de Methoden te zetten.

STAP 4: DE KLASSE BANKREKENINGFORM

Declareer binnen de BankrekeningForm twee Fields, voor de 2 bankrekeningen. Initialiseer deze twee Fields vervolgens in de Constructor van het formulier.

```
public partial class BankrekeningForm : Form
{
    // Fields
    private Bankrekening bankrekeningLinks;
    private Bankrekening bankrekeningRechts;

    // Constructor
```

NIVEAU

★★★☆☆

Figure 19-2 niveau

```
public BankrekeningForm()
{
    InitializeComponent();
    bankrekeningLinks = new Bankrekening("Duck, Dagobert");
    bankrekeningRechts = new Bankrekening("Gans, Gijs");
}
```

Maak nu de EventHandlers voor de knoppen aan. Weet je het nog? Door dubbel te klikken op de Button in het ontwerpscherf, wordt de standaard EventHandler (Click) automatisch aangemaakt. Vul de EventHandlers voor alle Buttons nu in om de gevraagde functionaliteit en de foutafhandeling te realiseren. Gebruik daarbij de methoden van de klasse Bankrekening die je al gemaakt hebt. Controleer op geldige invoer en vergeet ook niet de Labels met saldo-informatie bij te werken. Je kunt hiervoor de methode ToString gebruiken, waarbij je opgeeft dat je het saldo als valuta wilt.

Note Als je wilt dat een numerieke waarde als valuta weergegeven wordt dan kun je daarvoor de methode ToString() gebruiken. Je geeft dan als parameter een hoofdletter C mee om aan te duiden dat het currency (valuta) is:

```
double saldo = 120.55;
lblSaldo.Text = saldo.ToString("C");
```

Test als je klaar bent of alle functies en de foutafhandeling goed werken. Gebruik hiervoor ook de lijst van gewenste functionaliteit op de eerste pagina's van deze opdracht.

TryParse

Als je wilt controleren of een ingetypte tekst een geheel getal is, kun je gebruik maken van onderstaande methode.

```
bool int.TryParse(string text, out int result);
```

Zoals je ziet is bij de declaratie van de tweede parameter out vermeld. Dit betekent dat deze methode de waarde van result mag aanpassen. Bij het aanroepen van de methode moet ook out vermeld worden. De waarde van de parameter result kan na het aanroepen van TryParse veranderd zijn. Als true wordt teruggegeven, bevat tekst een heel getal en heeft result de waarde. Zo niet, dan is er iets anders ingegeven dan een geheel getal en bevat result geen geldige waarde. Let op: een negatief geheel getal is ook een geheel getal.

```
if (int.TryParse(txtEuroLinks.Text, out getal))
{
    // de invoer in txtEuroLinks is een geheel getal en
    // de waarde zit nu in de variabele ''getal.
}
```

Zie MSDN voor meer informatie.

19.2 Uitbreidingen

Voeg een ListBox op het scherm waarin de transacties worden weergegeven onder vermelding van datum, tijd, betrokken rekeningnummer(s) en bedrag. Naar keuze geef je alleen geslaagde of zowel

geslaagde als niet geslaagde transacties weer. In het laatste geval wordt er tevens vermeld of de transactie geslaagd is of niet.

CHAPTER 20

Challenge Invaders

Niveau	3 of 5
Leerdoelen	Class, constructor, object, field.
Vereiste voorkennis	Basiskennis over objecten.
Challenge Type	Realiseren.

20.1 Inleiding

In deze opdracht gaan we een game programmeren dat “Invaders!!” heet. In dit spel ben je de verdediger van de aarde, en je beschermt onze aarde tegen de binnenvallende wezens. Je dient te voorkomen dat de wezens op de aarde landen en een stad vernietigen, door ze met de muis aan te klikken.

Als je op een wezen schiet door het aan te klikken raakt het een leven kwijt en gaat het weer naar boven in het scherm om opnieuw te proberen op de aarde te gaan landen. Wezens die de onderkant van het scherm bereiken tellen als “geland” en vernietigen een stad. Als alle wezens al hun levens kwijt zijn wint de speler, als echter alle steden vernietigd worden winnen de wezens.

20.2 Opdracht

Programmeer het spel Invaders!! waarbij je programma aan de volgende eisen moet voldoen:

1. Je programma bevat een klasse Invader met private velden Positie en Levens. Positie is de y-coördinaat van de invader op het scherm en Levens is een getal dat begint op 5 (elke keer dat de gebruiker met de muis op een invader klikt wordt het veld Levens met één verlaagd).
2. De constructor zet de beginwaarde van Levens op 5.
3. Invaders kunnen bewegen (van boven naar beneden), dood gaan (als ze geen levens meer hebben) en ze moeten de mogelijkheid hebben om hun leven te verlagen (als je er op klikt). Programmeer dat netjes met methoden in de klasse Invader (en roep die methoden dan vanuit je form aan).
4. Maak in je form een array (array's worden behandeld bij FUN12 en pas je hier toe) van 6 Invader-objecten aan en gebruik die om in het form verder het spel af te programmeren.

CHAPTER 21

Challenge Auto Dagwaarde

Niveau	2 of 5
Leerdoelen	Class, property.
Vereiste voorkennis	Basiskennis over objecten.
Challenge Type	Realiseren.

Je bent als software engineer ingehuurd om een tool te programmeren waarmee de dagwaarde van auto's kan worden bepaald.

De dagwaarde van een auto wordt berekend met deze formule:

$(500000/\text{KM}) * \text{factor}$

Hierbij is KM de kilometerstand van de auto. Factor is een waarde die afhankelijk is van het brandstoftype:

- Factor = 100 indien het een benzine-auto is
- Factor = 150 indien het een dieselauto is.
- Factor = 90 indien het een LPG-auto is.

De user interface bestaat minimaal uit 3 radio buttons, een textbox en een knop (button) om de berekende dagwaarde te tonen.

Technische randvoorwaarden

De volgende technische requirements zijn van toepassing:

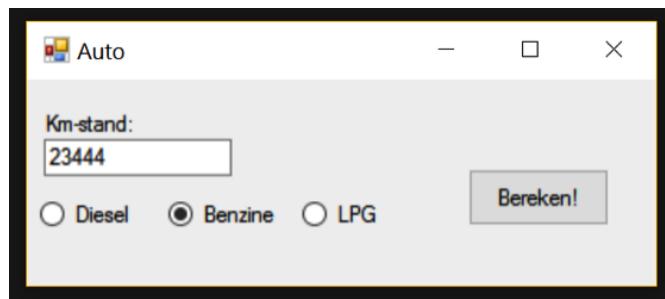


Figure 21-1 dagwaardeberekening

- Er dient een klasse Auto geprogrammeerd te worden met property's KmStand en Brandstofsoort.
- De klasse Auto dien ook een read-only property Dagwaarde te hebben
- Het form maakt gebruik van deze klasse: er staat geen code die berekeningen uitvoert in het form.

Challenge Exception Handling

Niveau	2 of 5
Leerdoelen	Exception handling.
Vereiste voorkennis	Basiskennis over objecten.
Challenge Type	Realiseren.

Als startmateriaal is het programma Naamgenerator beschikbaar. Open deze solution en bekijk de code eens. Het programma genereert een willekeurige naam uit een lijst van beschikbare namen en laat die naam op het scherm zien. De lijst van beschikbare namen staat in het bestand Naamen.txt. Het programma leest dit bestand uit.

22.1 Opdracht 1

Voeg C#-code toe die alle exceptions van het programma zoals bijvoorbeeld de `IndexOutOfRangeException` en exceptions die te maken hebben met bestanden/netwerk opvangen en een duidelijke foutmelding aan de gebruiker laten zien of (nog beter) de fout oplossen zonder dat de gebruiker het ziet.

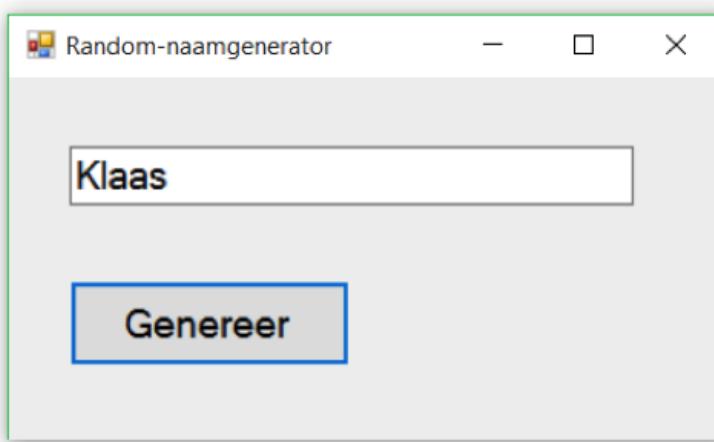


Figure 22-1 naamgenerator

22.2 Opdracht 2

Breidt het programma uit zodat er naar keuze 1 of 2 namen worden geselecteerd. De gebruiker moet van te voren kiezen of hij 2 namen of 1 naam wil zien. Voeg user interface controls toe, en vervang bestaande controls naar keuze.

Challenge Dino Game

Niveau	3 of 5
Leerdoelen	Analyse, class, enum, (pijltjes)toetsen, file read/write.
Vereiste voorkennis	Basiskennis over objecten.
Challenge Type	Realiseren.

23.1 De opdracht – Dino-spel

Maak een spel maken waarbij 2 dinos over het veld kunnen bewegen, één dino is groen, de ander is rood. De dino's zitten ieder in hun eigen kooi. Beide dino's hebben hun eigen knop om die dino te activeren je kan de dino dan besturen door middel van de pijltjestoetsen. Er verschijnen munten op het veld, en deze moeten door één van de dino's worden opgeraapt. Een score wordt bijgehouden.

Eisen:

1. Analyse/vooronderzoek: start een onderzoek op met de hoofdvraag Hoe kan ik een applicatie goed besturen met de pijltjestoetsen. Maak een proof-of-concept waaruit blijkt dat dit werkt.
2. Toon ook de high score op het scherm. Sla de high score op in een tekstbestand en lees de high score weer uit bij het opstarten van het spel.

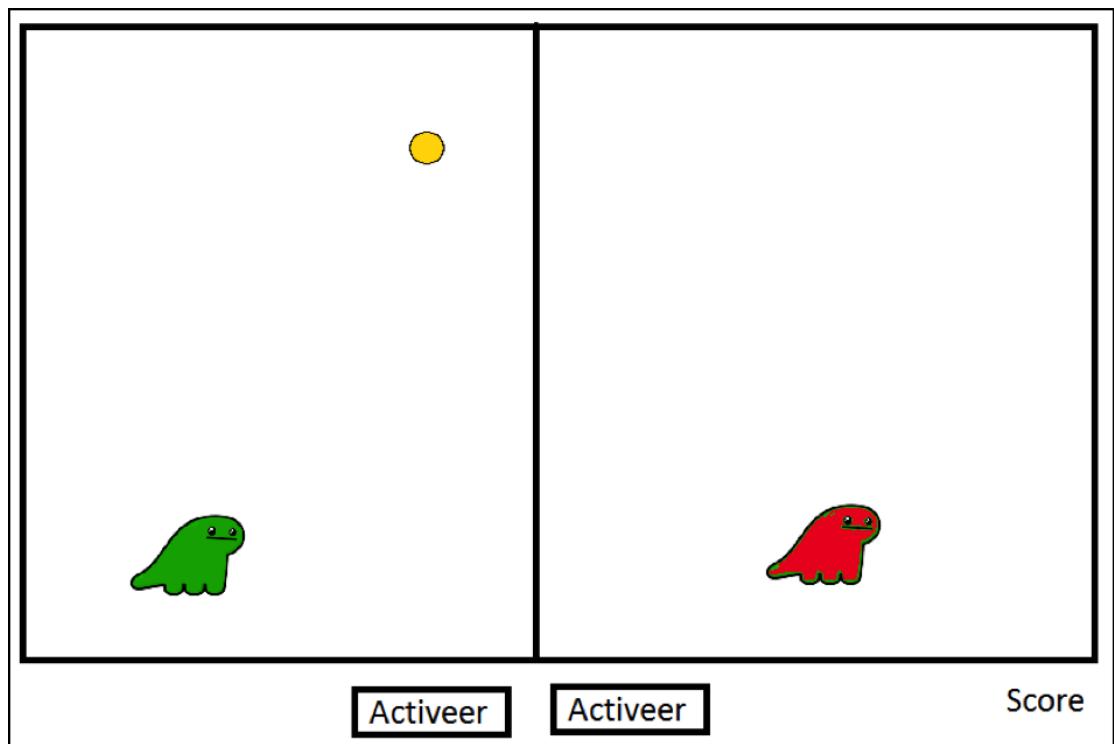


Figure 23-1 dino

CHAPTER 24

Challenge Cube Graphic

Niveau	3 of 5
Leerdoelen	Paint event, graphics.
Vereiste voorkennis	Basiskennis over objecten.
Challenge Type	Realiseren.

24.1 The Cube

Teken een kubus als wireframe in een panel. Als de gebruiker het form vergroot, verkleint en weer maximaliseert dan blijft de kubus zichtbaar.

Maak de voorste zijde van de kubus rood, dus vul het op met rood.

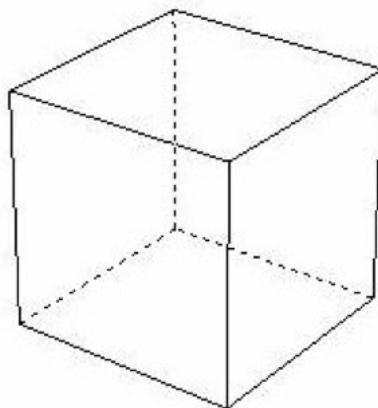


Figure 24-1 cube

CHAPTER 25

Challenge Ms PacMan

Datum	Week 11/12
Versie	1 - Marcel Veldhuizen
Leerdoelen	Paint event, graphics.
Vereiste voorkennis	Classes.
Challenge Type	Programming

25.1 De opdracht

Begin jaren tachtig was **Ms. Pac-man** een populaire game. Maak een App(lication) met een form dat helemaal zwart is en ga dan aan de slag met het tekenen van een aantal elementen uit Pac-man, om te oefenen met Graphics.

Teken op het veld een witte stip. Dit is een vierkant gevuld met witte kleur die bestaat uit 4 pixels.

Teken op het zwarte veld een grote pil. Een grote pil is vorm die op te vatten is als een 8-hoek of kan opgebouwd worden uit een aantal andere vormen. Zie figuur 25-2.

Teken een muur zoals die op het speelveld te zien zijn, bestaande uit een horizontale balk, met afgeronde hoeken. De rand van de balk is rood, de binnenkant is gevuld met een lichtrode/bijna roze kleur. Zie figuur 25-3

.

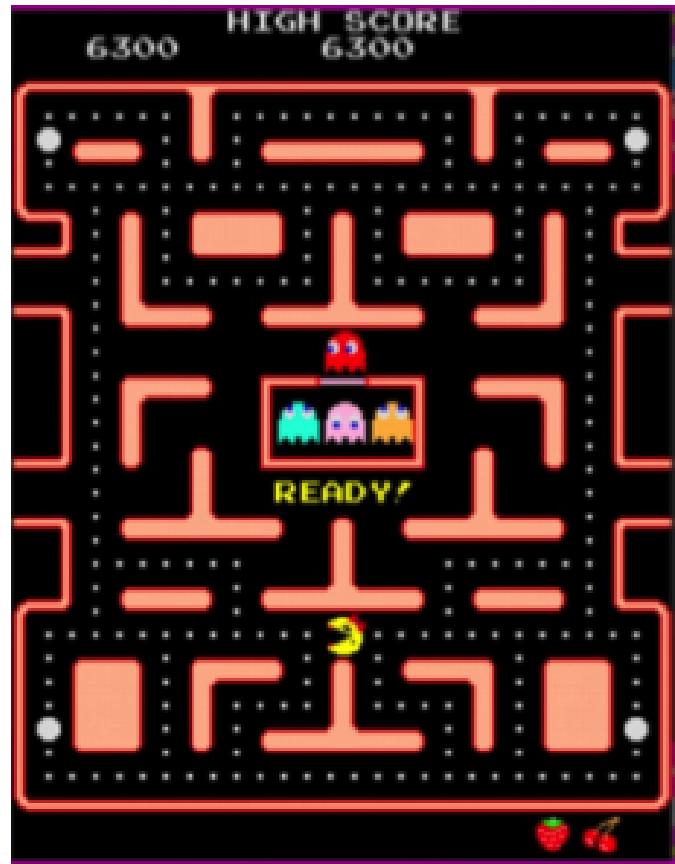


Figure 25-1 pacman

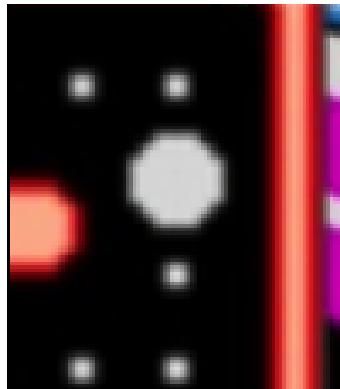


Figure 25-2 pil



Figure 25-3 muur

CHAPTER 26

Challenge Woordenzoeker

Versie	1 - Jan Oonk
Niveau	4 of 5.
Leerdoelen	Class, Property, Constructor, private/public, UI separation, algoritme, file read/write.
Vereiste voorkennis	Method, GUI, Basic Types, If.
Challenge Type	Programming, algoritme.

Op basis van een set woorden (uit een tekstbestand) een woordenzoeker puzzel van een opgegeven breedte x hoogte genereren. Woorden worden verstoopt zowel horizontaal (links-rechts al dan niet achtste voren) als verticaal (boven-beneden al dan niet achterste voren). Zorg ervoor dat de woorden binnen het speelveld blijven en niet "wrappen" naar de andere kant van het speelveld.

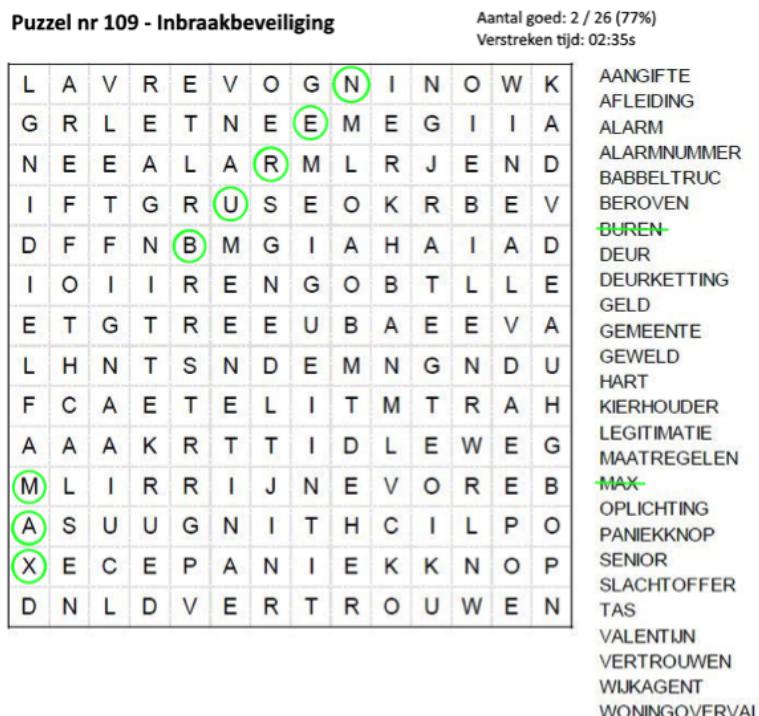


Figure 26-1 woordenzoeker

Daarna moet de speler de woorden kunnen zoeken. Een deel van het spelersscherm bestaat uit overzicht van woorden die gezocht moeten worden en ander deel is het speelveld van letters.

Speler kan woorden aanstrepen in het speelveld door letters te kiezen middels de linkerknop van de muis. Een letter die al geselecteerd is, wordt bij opnieuw aanklikken gedeselecteerd. De gekozen letters worden meteen automatisch gecontroleerd na aanklikken.

Letters kunnen alleen worden geselecteerd in dezelfde richting als de vorige geselecteerde letter(s). Als de geselecteerde letters als woord worden herkend in de lijst van verstopte woorden dan wordt deze doorgestreept.

Tijdens het selecteren van letters, worden oranje omcirkeld. Als de geselecteerde letters een nog te zoeken woord vormen worden de cirkels definitief groen.

Letters kunnen vaker worden geselecteerd en onderdeel zijn van meerdere woorden.

Laat een timer zien hoe lang de speler al aan het spelen is.

Om de letters in de vakjes te plaatsen kan je DrawText gebruiken of een *Sprite Generator* (gebruik monospace font als *Consolas,Regular,72*) gebruiken en deze plaatjes dynamisch inladen.

Leerdoelen OIS12

Zeker: Class, constructor, private/public, property, field, method, file handling, GDI graphics, enum, lijsten,

Hoogstwaarschijnlijk/goed mogelijk: method/constructor overloading, UML, exceptions, static, casting

Eventueel: XAML

Bronnen

Sprite Font Generator¹

Write Text on a Bitmap²

Bronnen benodigd bij de extra features

Sound Engine³

Database connectie e.d.⁴

Webclient⁵

How to: Request Data Using the WebRequest Class⁶

What difference is there between WebClient and HTTPWebRequest classes in .NET?⁷

JSON Parser⁸

Variatie / extra features:

Niveau - Feature

¹[https://www.scirra.com/forum\(sprite-font-generator-v3_t86546](https://www.scirra.com/forum(sprite-font-generator-v3_t86546)

²<https://stackoverflow.com/questions/6311545/c-sharp-write-text-on-bitmap>

³<https://www.ambiera.com/irrklang/downloads.html>

⁴<http://csharp-station.com/Tutorial/AdoDotNet>

⁵[https://msdn.microsoft.com/en-us/library/system.net.webclient\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.net.webclient(v=vs.110).aspx)

⁶<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/how-to-request-data-using-the-webrequest-class>

⁷<https://stackoverflow.com/questions/4988286/what-difference-is-there-between-webclient-and-httpwebrequest-classes-in-net>

⁸<https://www.newtonsoft.com/json>

- * - sla de highscores (is verstreken speeltijd) en de naam van de speler op in een bestand.
- * - Variatie toevoegen, via een menuoptie, om getallen te zoeken i.p.v. woorden.
- * - moeilijkheid van de te genereren puzzel is in te stellen via een start menu o.i.d., bijv. door woorden vaker achterste voren in de puzzel te zetten, diagonaal te plaatsen.
- * - Als alle woorden gevonden zijn dient de speler de overgebleven letters in de juiste volgorde te zetten om zo een woord of correcte zin te vormen.
- * - categoriseer de woorden in thema's, zodat verstopte woorden allemaal met elkaar te maken hebben bijv. categorie/thema Planten, Films o.i.d.
- * - De woorden, die in de te genereren puzzel worden verstopt, uit een database halen i.p.v. een bestand. Maak eerst een eenvoudig database model en ontwerp.
- * - Voeg leuke sounffects toe bij bepaalde events zoals correct woord, incorrect woord, letter selecteren, puzzel af e.d.
- * - Als op escape wordt gedrukt, worden de eventuele geselecteerde letters gedeselecteerd.
- * - 2e letter selectie manier inbouwen: letters kunnen ook worden geselecteerd door de linker-knop in te houden en te slepen. Alle letters worden geselecteerd tussen beginletter en muispositie. Bij loslaten wordt het woord, dat gevormd wordt door de selecteerde letters, gecontroleerd. Is het fout/nog niet goed dan kan in dezelfde richting opnieuw worden geselecteerd (door de normale manier of deze nieuwe manier), de vorige geselecteerde letters blijven dus actief (oranje).
- * - Als je letters selecteert door deze aan te klikken en je kiest de volgende letter niet in dezelfde richting en/of de letter is niet aangesloten met 1 van de zojuist eerdere geselecteerde letters dan worden al deze geselecteerde letters gedeselecteerd.
- ** - sla de highscores op op een centrale plek (file of database). Maak hiervoor een eenvoudige webservice.
- ** - spelers willen graag hun highscores op een eerlijke manier met elkaar vergelijken, dus zorg ervoor dat de puzzels op exact dezelfde manier aan de verschillende spelers wordt gepresenteerd, dus alle letters in een puzzel en verstopte woorden staan op exact dezelfde plek. Toon de highscores per puzzel. Geef hiervoor elke puzzel een unieke naam of id. Toon dit in een selectiemenu zodat je een bepaalde puzzel kan inladen.
- *** - Als snel blijkt de highscore webservice (zie ** uitbreiding) te zijn gehacked door script kiddies die de url hebben weten te achterhalen. Verzin een manier om de webservice te beveiligen, zodat er niet onrealistische of onterechte highscores kunnen worden verstuurd naar de webservice.

Challenge Super Galgje

Niveau	4 of 5.
Leerdoelen	Class, Property, Constructor, private/public, UI separation, algoritme.
Vereiste voorkennis	Method, GUI, Basic Types, If.
Challenge Type	Programming, algoritme.

27.1 Super-galgje

Galgje is een spel waarbij een speler het woord moet raden dan de computer in gedachten heeft.
Opdracht: schrijf galgje en maak gebruik van de object georiënteerde mogelijkheden van C#.

1. Te programmeren classes: *Woord* en *Form1* (form).
2. Te programmeren property in de class *Woord*: *AantalLetters*.
3. Te programmeren methode in de classe *Woord*: bool *IsGoed(string woord)*
4. Te programmeren classes: *Woord*, *SpelStatus*, *Form1*.
5. *Form1* heeft een referentie naar 1 *SpelStatus*-object en geen referentie naar *Woord*.
6. Te programmeren properties in de class *Woord*: *AantalLetters* (read-only property).
7. Te programmeren methoden in de class *Woord*: bool *IsGoed(string woord)*.
8. Te programmeren property in de class *SpelStatus*: *HetWoord* van het type *Woord* (dus NIET van het type *string*).
9. Verder kun je in de class *SpelStatus* methodes en/of properties toevoegen die de status van het spel zoals het aantal geraden letters bijhouden.

Eisen voor gevorderden:

1. Programmeer er nog een class *Speler* bij en zorg ervoor dat je met twee personen het spel tegen elkaar (tegen de computer) kunt spelen.
2. Class *Speler* heeft een property van het type *SpelStatus* en diverse methodes die jij zelf bedenkt. Het form krijgt 2 *Speler*-objecten en verder geen enkel ander object.

28

CHAPTER

Challenge File Handling

Niveau	5 of 5: integraal.
Leerdoelen	Class, Property, Constructor, private/public, UI separation, algoritme.
Vereiste voorkennis	Goed om kunnen gaan met classes en objecten en GUI separation.
Challenge Type	Programming.

Opdracht

Bestudeer de C#-klassen van het .NET-framework waar mee je kunt werken met bestanden, folders, mappen enz. C# kent allerlei methoden om tekstbestanden te lezen en te schrijven, bestanden te verplaatsen, folders aan te maken, enz. Werk vervolgens een aantal van onderstaande casussen uit om te oefenen met file handling en vraag feedback aan je docent:

De Opdracht

Kies in overleg met je docent alle of enkele van de onderstaande casussen uit om te oefenen met file handling.

Casus 0 - Analyse

Een klasse is te zien als een structuur in C# die een aantal methoden heeft die bij elkaar horen. Schrijf een kort document waar je in beschrijft welke functionaliteiten de volgende klassen in het .NET-framework hebben. Beschrijf wat je als C#-programmeur met deze klassen kunt doen, geef een paar korte codevoorbeelden.

- File
- Directory
- DirectoryInfo
- Path

Casus 1 - Tekstbestandzoeker

Schrijf een Windows Forms C#-programma dat voldoet aan de volgende requirements:

1. Een user interface met minimaal 2 listboxen.
2. Na het opstarten van de app wordt in listbox1 een lijst van alle folders op de root (C:) van je harde schijf getoond.

3. Als ik op een folder (item in de listbox) klik dan verschijnt er in de andere listbox een lijst van alle bestanden met extensie .TXT.

Casus 1a - For-lus

Breid de applicatie van casus 1 uit met een for-lus die alle bestanden die met de letter a beginnen NIET laat zien (skipt) in de tweede listbox.

Casus 2 - Tekstverwerker

Maak een applicatie waarmee tekstbestanden kunnen worden bewerkt. De minimale user interface is een tekstveld, een Opslaan-knop en een Openen-knop.

1. De gebruiker kan vanuit een Windows Forms user interface op een nette manier een tekstbestand selecteren vanuit zijn computer.
2. Na een druk op de Openen-knop wordt de volledige inhoud van het bestand dat de gebruiker heeft geselecteerd getoond in een multiline tekstveld. De gebruiker kan desgewenst de tekst in het veld wijzigen.
3. Na een druk op de Opslaan-knop wordt de tekst die op dat moment in het tekstveld staat opgeslagen in het eerder geselecteerde bestand.

Casus 3 - Word light

1. Maak casus 1 en casus 2.
2. Maak een derde applicatie die de functionaliteit van zowel casus 1 als casus 2 bevat. Maak hierbij gebruik van een tab-control waarbij de gebruiker op de eerste tab een bestand kan uitkiezen en op de tweede tab het bestand kan bewerken.

Casus 4 - Fruit-generator

1. Laat de applicatie een bestand met de naam "fruit.txt" aanmaken.
2. Laat de applicatie op 5 regels de teksten "banaan", "sinaasappel", "kiwi", "mandarijn" en "aardbei" wegschrijven. Open het bestand in Windows Kladblok om te kijken of je applicatie goed werkt.
3. Breidt de applicatie uit zodanig dat ze een melding geeft als het bestand "fruit.txt" al bestaat.

Casus 5 - Mijn computer

Maak een applicatie die alle schijven van je computer laat zien (zoals Windows Verkenner dat kan) in een eenvoudige listbox. Zie 28-1

Casus 5a

Voeg aan de Mijn Computer-applicatie de volgende functionaliteiten toe:

- Mogelijkheid om een bestand te kopiëren naar een andere locatie
- Mogelijkheid om een bestand te hernoemen

Casus 5b (verdieping)

Toon tijdens het kopiëren van een bestand (zie Casus 5a) een progress bar die aangeeft hoe lang de kopieeractie nog duurt.

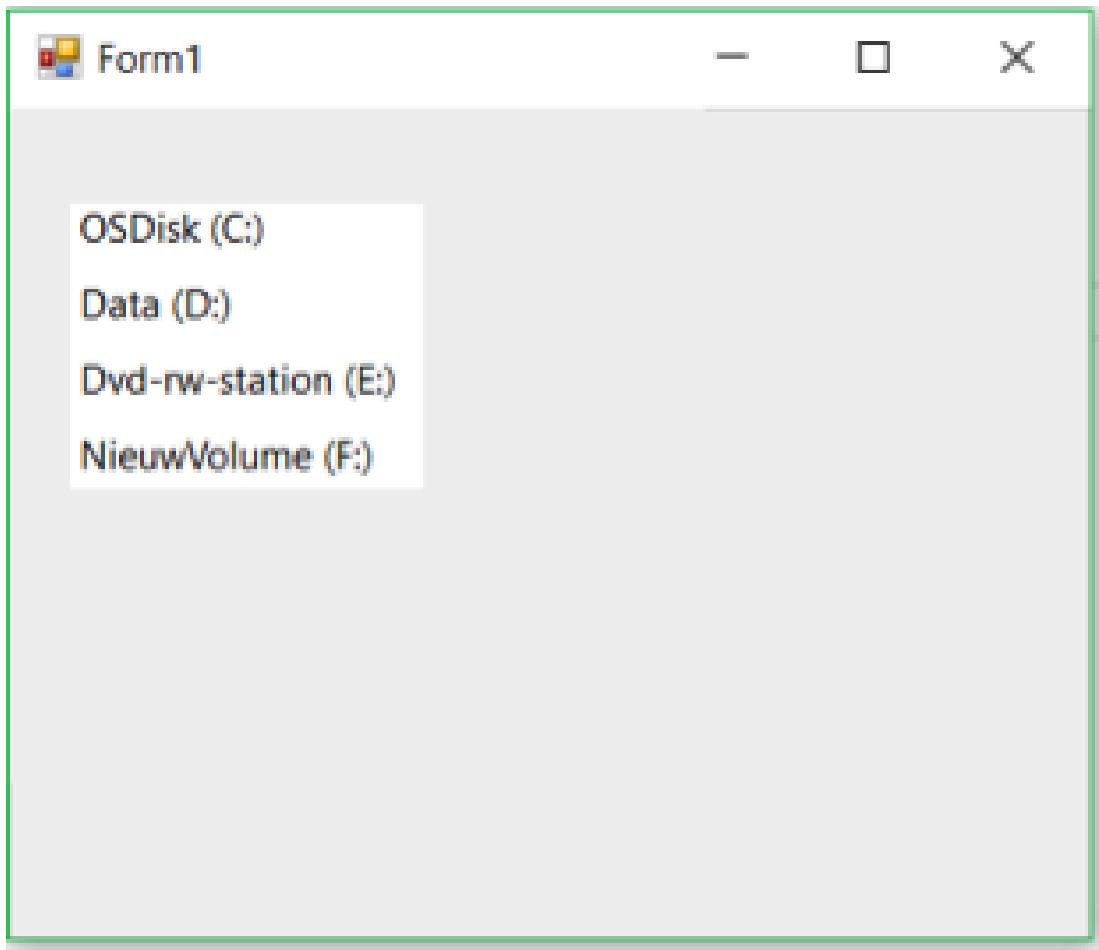


Figure 28-1 Mijn Computer.

Casus 6 - Zoek in bestand

Maak een applicatie die een willekeurig door de gebruiker ingevuld woord kan opzoeken in een bestand. Toon het regelnummer in het bestand waar het woord voorkomt op het scherm.

29

CHAPTER

Challenge Iedereen kan Schilderen

Niveau	4 of 5: integraal.
Leerdoelen	Class, Property, Constructor, private/public, GUI separation, Paint Event, Graphics, file read.
Vereiste voorkennis	Goed om kunnen gaan met classes en objecten en GUI separation.
Challenge Type	Programming.

29.1 Leerdoelen

Met deze opdracht kun je laten zien dat je File Handling, Exception Handling en Graphics binnen één applicatie kunt programmeren.

- Je kunt vanuit een specificatie een programma schrijven dat werkt met de diverse functies van de Windows GDI (graphics).
- Je moet vanuit een specificatie een C#-programma kunnen schrijven waarmee tekstbestanden kunnen worden ingelezen en worden geschreven.
- Je moet op een correcte manier exceptions in je C#-programma kunnen afhandelen.

29.2 Casus 1 - Iedereen kan schilderen

Schrijf een Windows Forms C#-programma dat een tekstbestand uitleest waar graphics-commando's in staan. Het programma gaat vervolgens al die commando's uitvoeren. Een commando in het tekstbestand is een instructie waarmee jouw programma cirkels en lijnen mee kan tekenen.

Voorbeeld van de inhoud van het tekstbestand:

```
cirkelR  
lijn  
lijn  
cirkelR  
cirkelB
```

Als de gebruiker met jouw programma bovenstaand bestand inleest dan zullen er een rode cirkel, een lijn, nog een lijn, nog een rode cirkel en een blauwe cirkel worden getekend.

De lijnen en cirkels verschijnen op willekeurige posities op het scherm.

Mogelijke commando's: lijn, cirkelR, cirkelB.

Functionele requirements

- De gebruiker moet met een standaard Windows dialoog (tip: gebruik de OpenFileDialog uit de Toolbox van Visual Studio) een tekstbestand met de commando's kunnen openen.
- De lijnen en cirkels verschijnen op willekeurige posities op het scherm.
- De C#-code maakt gebruik van exceptiehandling om foutmeldingen met bestanden te voorkomen.

29.3 Casus 1a - Advanced painter

Breid het programma van casus 1 uit zodat de gebruiker:

- Meerdere bestanden tegelijk kan openen.
- Meer commando's kan gebruiken, bijvoorbeeld om groene driehoeken en teksten te tekenen.

29.4 Casus 2 - Snake

Snake is een tekenprogramma waarmee op een snelle manier lijnen kunnen worden getrokken.

Schrijf een Windows Forms C#-programma dat voldoet aan de volgende requirements:

- Als de gebruiker op de muisknop klikt dan wordt de cursor-positie (x,y-coordinaat) van dat moment opgeslagen in twee variabelen x en y.
- Als de gebruiker de muis beweegt en dan op een andere positie nog een keer klikt dan wordt er een lijn getrokken van het eerste punt naar het tweede punt.

29.5 Casus 2a - Snake Pro

Breidt het programma van casus 2 uit met de volgende functionaliteit:

- Mogelijkheid om vierkanten en/of rechthoeken te tekenen. De gebruiker selecteert met een radiobutton van te voren wat hij wil tekenen.

CHAPTER 30

External Challenges

Op internet zijn veel sites te vinden met uitdagend oefenmateriaal op programmeergebied.

Codingame

Kijk bijvoorbeeld naar [codingame.com](https://www.codingame.com/home)¹: Een platform waarbij je allerlei opdrachten in C# kunt maken gesorteerd op moeilijkheidsgraad en onderwerp!

Code Golf:

Een ander voorbeeld: codegolf.stackexchange.com²

ProjectEuler:

Een site waar allerlei (vaak wiskundige) programmeeruitdagingen op staan is projecteuler.net³. Door softwaredocenten is Euler-project 54 (genaamd *poker hands*) al genoemd als een voorbeeld van een leuke uitdaging.

Dwitter.com

<https://dwitter.com>

¹<https://www.codingame.com/home>

²<http://codegolf.stackexchange.com/>

³<https://projecteuler.net/>

Create your own Challenge

Een hele goede manier om te leren en te kijken of je een onderwerp helemaal snapt is het zelf bedenken van een opdracht.

Puzzel of Spel

Neem een spel of puzzel in gedachten en bedenk hoe je dit kunt implementeren. Enkele voorbeelden hiervan zijn *boter, kaas en eieren*, *tetris*, *sudoku* of *mastermind*, *snake* of *tron*, *2048*.

Maak lesmateriaal

Je helpt ook je medestudenten als je zelf lesmateriaal ontwikkelt voor een nieuwe programmeeropdracht. Schrijf zelf je eigen opdracht en lever deze in bij de docent. Dan adopteren we jouw idee en leeft jouw eigen opdracht-idee misschien wel verder als officieel lesmateriaal.

32

CHAPTER

Challenge Uitbreiding BankRekening (UWP)

[File -> New -> Project -> Windows Universal -> Blank Universal App.

Maak je bankrekening in deze UWP App. Kopieer en plak je Bankrekening c# class in je nieuwe UWP app. Dus je gebruikt je al eerder gemaakte class van die WinForms opdracht.

Als je de bankrekening class "goed" gemaakt hebt dan zit er geen referenties in naar methodes en dingen die niet meer bestaan en hoef je alleen de buttons en labels te slepen.

Je komt er achter dat niet alles een naam heeft en dingen zoals MessageBox.Show niet meer bestaan in UWP.

Challenge Windows Phone

33.1 Voorkomende leerdoelen

file handling, exception, private/public, casting, list en foreach, UI separation.

33.2 Vereiste voorkennis

OIS11.

Als je bij OIS11 nog niet hebt gewerkt aan een Windows Phone-app dan kun je dat nu mooi doen! Doel: het programmeren van een Windows Phone app die aan de hand van de GPS-locatie laat zien of je thuis bent of op school bent. Hierbij kun je stap-voor-stap te werk gaan. Let op dat je alle stappen rustig en beheerst doorloopt, je hebt geen haast. Het doel is dat je iets leert. Het einddoel is niet een werkende app (hoewel dat wel mooi zou zijn natuurlijk). 1. Onderzoek wat je nodig hebt voor de ontwikkeling van Windwos Phone apps (bijv. juiste versie van Windows en juiste versie van Visual Studio en ondersteunende tools om een Windows Phone emulator te kunnen draaien). 2. Installeer de complete ontwikkelingomgeving en test of de Windows Phone emulator werkt. Maak de emulator werkend. Tip: je kunt ook bij de ISSD een echt device (Windows telefoon of tablet) lenen natuurlijk. 3. Onderzoek hoe je in Windows Phone de GPS (positie) van de gebruiker uit kunt lezen en ga op zoek naar een algoritme waarmee je kunt bepalen of je dichter bij huis of bij school bent. 4. Programmeer de app.

Uitbreiding 1. Breidt je werkende app uit met een Bing-maps kaart of een Google-maps kaart. Hiervoor moet je eerst onderzoeken wat je daarvoro nodig hebt, zoals een developer API-key en misschien een externe library. Laat op de kaart met een drietal markers het volgende zien: a. Een marker die de huidige GPS-positie van de telefoon aangeeft. b. Een marker die de school aangeeft (Rachelsmolen 1). c. Een marker die je thuislocatie aangeeft.

CHAPTER 34

Advanced Challenge Memory in Unity 3D

Dit is een complexere opdracht. Het is een voorbeeld van wat je zou kunnen doen als je een hoog punt wilt halen. Hierbij hoort wel dat je dingen zelf zult moeten uitzoeken.

In deze opdracht ga je aan de slag met Unity. Onder anderen komen er Arrays, lists en classes aan bod. Unity werkt iets anders dan je misschien gewent bent.

Het startmateriaal kun je vinden achter de url: startmateriaal¹

Kijk eens naar:

Instantiate²

Coroutines³

Unity heeft ook de volledige scripting reference ingebouwd. Als je in de editor op F1 druk open je deze. Op internet kun je ook heel er veel vinden, mede door Unity answers, waar gebruikers vragen kunnen stellen die dan door de community worden beantwoord. Op youtube staan veel verschillende tutorials gemaakt door gebruikers. Ook Unity zelf heeft veel gratis training materiaal, ook doen zij live trainingen die daarna terug te kijken zijn: <http://unity3d.com/learn>

Het Unity project

Het project is gemaakt in Unity versie 5.1.3f1. Als je de nieuwste versie hebt gedownload zit je altijd goed. In het project vind je een volledig werkend voorbeeld. Verder vind je een map met zelf doen. Daarin zijn al een aantal dingen opgezet voor je. In de code vind je comments met alle stappen die je moet nemen. In het voorbeeld kun je zien hoe je dit zou kunnen aanpakken.

Veel succes!

¹https://github.com/coentjo/softwarelessons/raw/master/Challenge_Starting_Material/Challenge_start_AdvancedMemory_in_Unity3D.zip

²<http://docs.unity3d.com/ScriptReference/Object.Instantiate.html>

³<http://docs.unity3d.com/Manual/Coroutines.html>

Part IV

Bijlagen

Visual Studio installeren

Visual Studio 2017 (afgekort: VS) kun je op twee manieren legaal downloaden:

Manier 1

Je zoek op internet naar "Visual Studio 2017 community". Je komt dan bij de pagina van Microsoft uit waar je de gratis versie van VS kunt installeren. Deze noemen ze de Community Edition. Kies bij het installeren voor ".NET desktop development". Deze versie is voor het startsemester voldoende. Maar....

Manier 2

Er is ook een versie met veel meer opties en handigheden. Deze heet Visual Studio 2017 Enterprise (betaalde versie). Voor studenten van FHICT zijn er licenties voor de versie: Ga naar het student-plein¹ op de FHICT-portal². Klik daarna op Microsoft Imagine(DreamSpark)³. Je komt dan uit in een webwinkel waar alles 0 euro kost. Zoek daar op "Visual Studio Enterprise 2017". Let op dat je enterprise er bij tikt, anders laat de webwinkel eerst de community edition zien. Met deze enterprise versie kun je heel je opleiding vooruit.

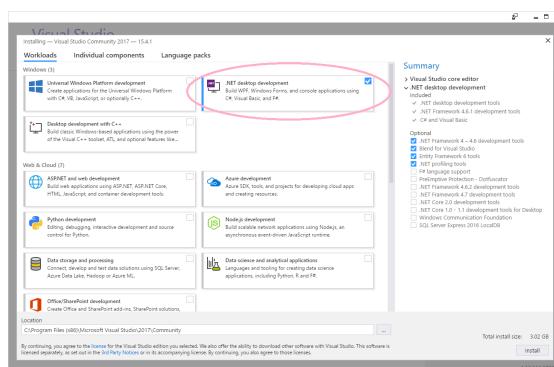


Figure 35-1 VSo20-components

¹<https://portal.fhict.nl/Studentenplein/SitePages/Home.aspx>

²<https://portal.fhict.nl>

³<https://apps.fhict.nl/OffCampusSSO/Dreamspark.aspx>

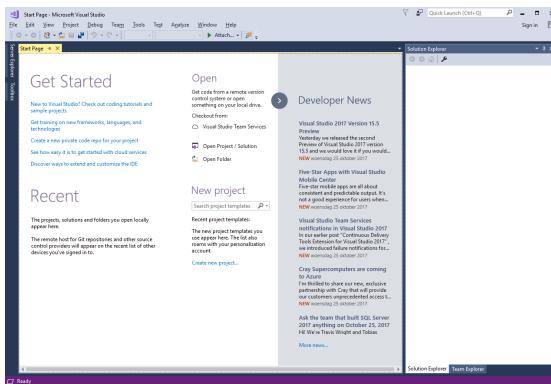


Figure 35-2 Dit is Visual Studio

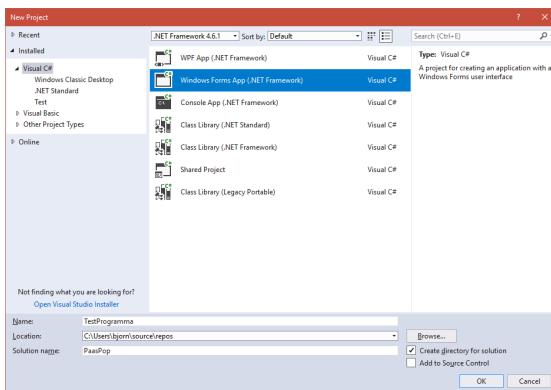


Figure 35-3 Dit is Visual Studio

Heb je een Mac?

Voor Apple laptop gebruikers is het mogelijk om VS 2017 op te starten in een Virtual Machine(VM) of in een bootcamp-omgeving. VS 2017 in een VM zal niet snel aanvoelen. Bootcamp is de beste oplossing. Zie <https://support.apple.com/nl-nl/boot-camp>

Voor zowel bootcamp alsmede VM oplossing heb je extra software nodig. Deze kun je gratis vinden in de webshop die gelinkt is in stap 2 hierboven.

Let op: er is een VS versie die ook werkt op een Apple laptop. Maar die versie is niet uitgebracht in de configuratie die we nu nodig hebben voor deze course. Gebruik die dus niet.

Je eerste programma

Wil je weten of je VS werkt? Start Visual Studio op. We maken eerst een nieuw project aan en gaan daarna C# sourcecode intikken. Vervolgens starten we de C# sourcecode op. Je eerste programma!

Als je dit ziet na de installatie, dan ben je startklaar voor de volgende stap.

Windows Forms App C#

Kies in het menu 'File' voor 'New' en dan 'Project'. Zie New Project.

Zorg ervoor dat je het blauwe deel exact hetzelfde laat zien als bij jou op het scherm. Sleutelwoorden: "Visual C#" staat links geselecteerd en er staat "Windows Forms App (.NET Framework) Visual C#". Geef het project een zelfgekozen naam bij "Name" en klik vervolgens op "OK".

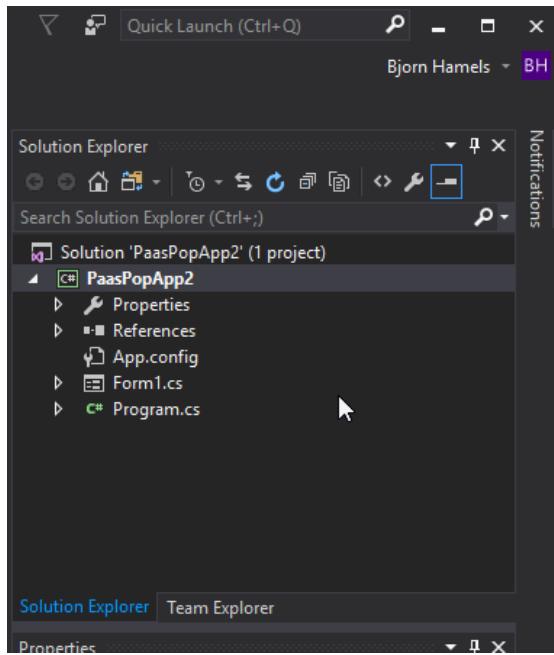


Figure 35-4 View Code

```

Form1.cs ✘ ✘ Form1.cs[Design]
WindowsFormsApp2 WindowsFormsApp2.Form1
1 用 System;
2 用 System.Collections.Generic;
3 用 System.ComponentModel;
4 用 System.Data;
5 用 System.Drawing;
6 用 System.Linq;
7 用 System.Text;
8 用 System.Threading.Tasks;
9 用 System.Windows.Forms;
10
11 空 WindowsFormsApp2
12 {
13     公共 partial class Form1 : Form
14     {
15         公共 Form1()
16         {
17             InitializeComponent();
18             MessageBox.Show(System.Text.Encoding.UTF8.GetString(System.Convert.FromBase64String("SGVsbG8gV29ybGQh")));
19         }
20     }
21 }
22

```

Figure 35-5 password code

Tikken C# Sourcecode

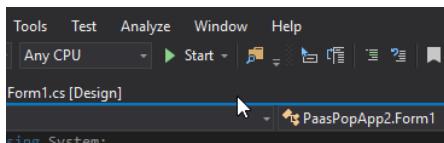
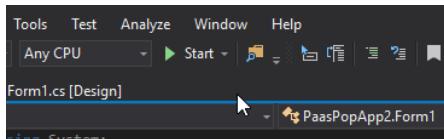
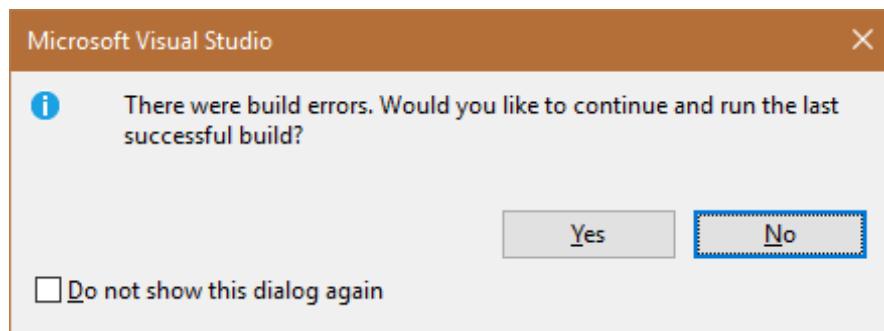
Om bij de source code van je eerste (lege) programma te komen moeten we die zichtbaar maken. Klik rechts op 'Form1' en kies 'View Code F7'.

Je ziet nu in het midden C# source code. Kopieer de regel hieronder en plak die onder de regel die er al staat. Maak het zo dat je scherm er hetzelfde uitziet als hieronder. Het gaat om het toevoegen van alleen deze regel. Sommige namen in de regels erboven en eronder heten wat anders. Dat komt omdat in het voorbeeldprogramma de namen misschien net wat anders gekozen zijn. Dat is niet erg.

```
[ MessageBox.Show(System.Text.Encoding.UTF8.GetString(System.Convert.FromBase64String("SGVsbG8gV29ybGQh")));
```

Wezenlijk voeg je dus alleen maar "MessageBox..."-regel toe onder de "InitializeComponent()"-regel.

Deze regel bevat een geheime code die een boodschap aan je laat zien bij het opstarten.

**Figure 35-6** run**Figure 35-7** stop**Figure 35-8** build error

Opstarten eerste programma

Nadat je C# sourcecode hebt getikt moet je Visual Studio de opdracht geven om het programma te maken (compileren) en uit te voeren (runnen). Dat doe je met de knop "Start". Je krijgt na het uitvoeren een boodschap te zien.

Het valt je op dat Visual Studio twee "gezichten" heeft. Een bewerksmodus en een uitvoeringsmodus.

- De bewerksmodus gebruiken we het meest. Daar kun je knoppen programmeren en C# sourcecode tikken. Visual Studio start op in deze modus en in deze modus hebben we net die regel C# sourcecode toegevoegd. In deze modus kunnen we ons nieuwe programma opstarten met "Start".
- Nadat je op "Start" klikt verspringen alle icoontjes. Soms blijft je C# sourcecode nog staan, soms niet. Je ziet Visual Studio je programma opstarten en grafieken tekenen van je CPU/Memory. Visual Studio is in deze modus bezig met het uitvoeren van je programma. Klik op het stop-blokje (zie afbeelding hierboven) om te stoppen met het uitvoeren en terug te gaan naar de bewerksmodus.

Het werkt niet?

Nu kan het gebeuren dat je een tiffout hebt gemaakt in je C# sourcecode. Dat laat Visual Studio zien met dit scherm:

Kies altijd voor "No"! Dan kun je terug naar je Visual Studio om te kijken waar je tiffout zit.

Tiffouten geeft Visual Studio aan met rode kringeltjes onder de woorden. Net zoals bij Word. Deze zijn soms moeilijk te begrijpen. Haal dan de regel weg en tik hem opnieuw. Je kunt ook het pro-

A screenshot of a code editor showing a tooltip. The code is in C# and includes a line that attempts to use the `TikFout` class from the `System` namespace. A tooltip box appears at the cursor position, containing the text: "The type or namespace name 'TikFout' does not exist in the namespace 'System' (are you missing an assembly reference?)". Below the tooltip, there is a link to "Show potential fixes (Alt+Enter or Ctrl+.)".

```
11  namespace PaasPopApp2
12  {
13      public partial class Form1 : Form
14      {
15          public Form1()
16          {
17              InitializeComponent();
18              MessageBox.Show(System.TikFout.Encoding.UTF8.GetString(System.Convert.FromBase64String("UGFhc3BvcA==")));
19          }
20      }
21  }
22
```

Figure 35-9 tikfout

gramma afsluiten en terug gaan naar een laatst werkende versie. (Niet opslaan.)

Hoe verder?

Probeer de designer te openen van Form1 (rechtsklik weer op Form1, net als bij het openen van de code). Sleep wat knoppen (buttons), tekst (labels), en dergelijke op je Form1. Pas de kleur en tekst aan. Maak er iets moois van!

Als je een knop dubbelklikt kun je deze code gebruiken om neer te zetten dat dan zichtbaar wordt als je op de knop drukt. (Plak deze code op de lege regel tussen de { en }).

```
[ MessageBox.Show("Dank je wel voor het klikken!");
```


Coding Guidelines

In elke programmeertaal zijn er afspraken over hoe code er uit ziet. Veel teams maken hier extra afspraken over.

1. Belangrijkste regel: *Leesbaarheid en Onderhoudbaarheid* van de code staan voorop! Op elke andere regel kan wel een uitzondering verzonnen worden, een geval waarin het leesbaardere of onderhoudbardere code oplevert als je er vanaf wijkt. Anderzijds is het sowieso eerst de *Regelen der Kunst* eerst goed te leren voor je er van af wijkt!
2. De naam van een Class wordt met een hoofdletter geschreven. Een Class name is normaal in het *enkelvoud*.
3. De naam van een Methode wordt in C# met een hoofdletter geschreven. Sommige teams maken afspraken over de volgorde van de Fields, Constructors, Methods binnen een Class, maar aangezien je in Visual Studio met F12 naar de declaratie van een Method, Constructor of Variable kunt springen maakt het niet uit.
4. De naam van een locale variabele wordt met een kleine letter geschreven.
5. De naam van een Property wordt in C# met een hoofdletter geschreven.
6. Gebruik een duidelijke inspringing. Hoe je de accolades en spaties en tabs zet kan op verschillende manieren: de precieze manier kan van team tot team verschillen en veel editors kunnen dit voor je doen. Als je het handmatig doet, doe het dan wel altijd op dezelfde manier.
7. Wen je aan alles meteen een goede naam te geven.

Programmer Search Scheme

Zie figuur Programmer Search Scheme. Vanwege het hoge little-donkey-bridge-gehalte is ervoor gekozen de termen en de afkorting (UPEAVL) in het Engels te houden.

Hieronder een bespreking *per stap*.

37.1 Understand

To solve a problem you must first Understand what the problem is!

- What is my starting point? An error? An exception? A bug? A feature on a wishlist?
- Snap ik het probleem waarvoor ik een oplossing zoek?
- Ken ik alle relevante begrippen?

37.2 (Create) Plan

Maak een plannetje:

- Waar ga ik naar zoeken?
- Hoe?
- Welke zoektermen?
- Welke zoekmachine?

37.3 Execute Plan

Voer plan uit:

- Gebruik de zoekmachine volgens plan.
- Selecteer en lees.

37.4 Adjust

- Vernoem je bronnen.
- Pas aan zodat je het resultaat snapt.

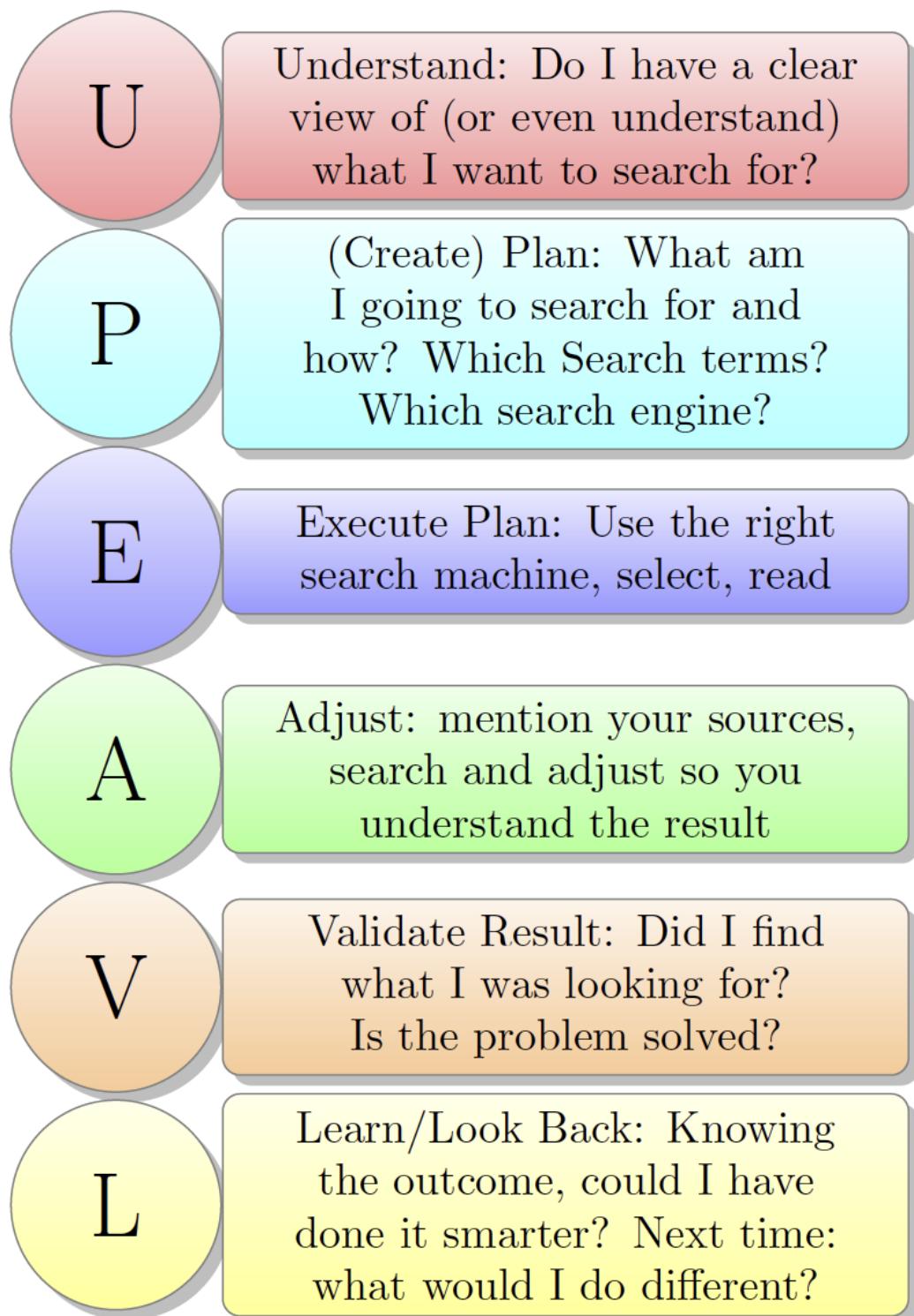


Figure 37-1 Programmer Search Scheme

37.5 Validate Result

Controleer het resultaat:

- Heb ik gevonden wat ik zocht?
- Is het probleem hiermee opgelost?
- Is de oplossing (altijd) correct of zijn er gevallen waar het niet voor werkt?

37.6 Learn/Look back

Leer:

- Wat heb ik geleerd?
- Nu je een antwoord kent: had je dit sneller, makkelijker, eleganter kunnen vinden? Heb je op het goede gezocht?
- Stel ik kom nog een keer een vergelijkbaar probleem tegen, kan ik dan sneller tot een oplossing komen?
- Moet ik iets ervan vastleggen? Zoja, waar?

Op <https://i889241.iris.fhict.nl/search/ProgrammerSearchScheme.pdf> is een wat uitgebreider document te vinden met voorbeeld.

Het idee komt in essentie van *How to solve it?* van de didacticus Polya. Het is begonnen als zogenaamd 100-uren-onderzoek van Coen bij FHICT. Veel collega's hebben feedback gegeven om te ontwikkelen en beter te maken. Hierbij werden ook een aantal alternatieven genoemd, zoals

bronnen

<http://www.rba.co.uk/wordpress/2016/09/19/essential-non-google-search-tools-for-researchers-top-tips/>

<https://decorrespondent.nl/5972/juist-nu-je-alles-kunt-googelen-moet-onderwijs-over-kennis-gaan/759327061724-15e0e58d>

https://en.wikipedia.org/wiki/Comparison_of_web_search_engines

<https://www.ghacks.net/2016/09/09/duckduckgo-programmers-search-engine/>

<http://html.com/blog/specialty-search-engines/>

http://www.academia.edu/34290333/Kritisch_zoeken_denken_en_evalueren_informatievaardigheden_als_21st_century_skill <http://bit.ly/MixedExtras>

External Resources

Enkele alternatieve bronnen:

Het youtube-kanaal van FHICT-docent Wilrik

YouTube Wilrik¹

BookBoon

BookBoon²

Brackeys

Brackeys³

The New Boston

Je kunt ook kijken naar De video-tutorials van The New Boston⁴

Hier de indeling:

¹<https://www.youtube.com/watch?v=luStUzWuPrw>

²<http://bookboon.com/en/object-oriented-programming-using-c-sharp-ebook>

³<https://www.youtube.com/watch?v=pSiLHe2uZ2w&list=PLPV2Kylb3jR6ZkG8gZwJYSjnXxmfPAI51>

⁴<https://thenewboston.com/videos.php?cat=15>

1	Visual Studio installeren Week 1
2	Form properties Week 2
3	MessageBox ois11 Week 1
4	Variabelen ois11 Week 2
5	Form properties in C# aanpassen ois11 Week 2
6	If-statement ois11 Week 3
7	If-statement ois11 Week 3
8	If-statement ois11 Week 3
9	Switch ois11
10	Rekenen in C# ois11 Week 2
11	Array's fun12 Week 3
12	Lists fun12 Week 4
13	For en foreach ois11-wk4 (for) fun12-Week 5 (foreach)
14	Do en do-while ois11-Week 4 (extra)
15	Try catch Leerdoel ois12
16	Methodes ois11-Week 6
17	Methodes ois11-Week 7
18	Continue and break
19	Namespace en class: fun12_Class
20	Constructors ois12 en fun12: Constr.
21	Private / public ois12 en fun12: Private / public
22	Overload / enums ois12:Enums fun12:ToString
23	Properties ois12:Properties
24	Exceptions ois12: Exceptions

MSDN tutorial

MSDN How do I Learn C# tutorials (Engelstalig)⁵

Bruikbaarheid: *

Toelichting: Enkele gedetailleerde walkthroughs. Aanrader als je al bekend bent met programmeren in een andere programmeertaal.

techzine tutorial

Les 1: Beginnen met C# (Nederlandstalig)⁶

Bruikbaarheid: ****

Toelichting: Uitleg over het maken van een programma aan de hand van een voorbeeldprogramma dat telkens een stukje wordt uitgebreid. Het gebruik van variabelen, FOR en WHILE lus worden uitgelegd. Let op: in deze tutorial wordt een Console applicatie gemaakt, dit is iets anders dan een Form applicatie.

Webbrowser tutorial

Zelf een webbrowser maken (Nederlandstalig)⁷

Bruikbaarheid: *

Toelichting: Tutorial waarin een webbrowser gebouwd wordt. Weinig toelichting op wat er gebeurt maar wel een leuk eindresultaat. Deze tutorial is vanaf lesweek 3 redelijk goed te maken.

⁵<http://msdn.microsoft.com/en-us/vcsharp/aa336766.aspx>

⁶<http://www.techzine.nl/tutorials/358/3/c-les-1-beginnen-met-c-de-eerste-stapjes.html>

⁷http://www.sitemasters.be/tutorials/17/1/564/CSharp.NET/CSharp_Zelf_een_WebBrowser_maken

Blackwasp tutorial

BlackWasp⁸

Bruikbaarheid: ***

Toelichting: Verzameling tutorials en artikelen (Engelstalig).

38.1 Online C-sharp boeken

C# Station Tutorial (Engelstalig)⁹ Bruikbaarheid: *** Toelichting: Uitleg over de basis onderdelen van C# zoals expressies, typen, variabelen en controlestructuren. De "lessons" 1 t/m 5 zijn interessant voor OIS.

C# Yellow Book (Engelstalig)¹⁰ Bruikbaarheid: **

Toelichting: Compleet boek over programmer constructies en onderwerpen. Wellicht handig als naslagwerk, niet geschikt als leerboek voor OIS. Let op: in deze tutorial wordt uitgegaan van een Console applicatie als beginpunt, dit is iets anders dan een Form applicatie.

⁸<http://www.blackwasp.co.uk/>

⁹<http://csharp-station.com/>

¹⁰<http://www.robmiles.com/c-yellow-book/Rob%20Miles%20CSharp%20Yellow%20Book%202010.pdf>

Programma-zip inleveren in Canvas

Als je gemaakte Visual Studio solutions wil inleveren in Canvas, hou dan rekening met het volgende:

- Laat Visual Studio tijdens het programmeren alle files opslaan op de door Visual Studio voorgestelde plek. (ga dus *niet* losse files met behulp van save as op andere plekken opslaan).
- Het gevolg hiervan is dat alle files van een solution in 1 directory bij elkaar staan.
- De default plek waar Visual Studio solutions opslaat is Documents\Visual Studio(+versienummer)\Projects
- Als je 1 solution wilt inleveren kun je de directory van die solution zippen door in het *rechtermuisknop* van de directory send to\Compressed (zipped) Folder te kiezen.
- Als je meerdere solutions in 1 zip wilt inleveren kun je er meerdere selecteren (met *ctrl* of *shift* ingedrukt en dan de directories aan te klikken), ook weer met hetzelfde *rechtermuismenu* te zippen.
- Gebruik in Canvas de Submit-button om de zip in te leveren.

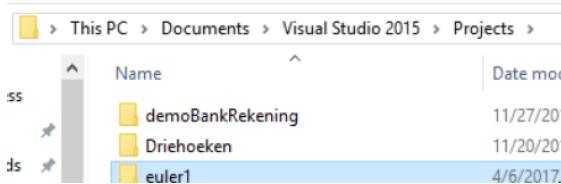


Figure 39-1 default project directory van Visual Studio



Figure 39-2 default project directory van Visual Studio

Handige sneltoetsen en opties in Visual Studio

Note Als in een sneltoets combinatie een komma staat, dan moet eerst het deel voor de komma worden ingedrukt, dan laat je de toetsen los en druk je daarna de letter in die na de komma staat.

Sneltoets	Menu	Toelichting
CTRL-W, X	View Toolbox	Maakt het Toolbox scherm met alle visuele objecten zichtbaar.
CTRL-W, P	View Properties	Window Scherm met de eigenschappen van visuele objecten tonen. Hier vind je ook de events van de visuele objecten.
CTRL-W, E	View Error List	Toont het scherm met de eventuele fouten die in je code gevonden zijn (ververs het scherm door je project opnieuw te build-en met de F6 knop, zie hieronder). Dubbelklikken op een error navigeert naar de plaats in je code waar de fout is gevonden. Tip: bekijk eerst de eerste fout, andere fouten kunnen hier een gevolg van zijn.
F6	Build Build Solution	Controleert je code op fouten en bouwt vervolgens een uitvoerbaar bestand.
F5	Debug Start Debugging	Build je project (zie F6 hierboven) als dat nog niet gebeurd is, en voert het uitvoerbaar bestand vervolgens uit. Dit doet hetzelfde als een klik op de knop met het groene pijltje (playknop).
Shift-F5		Stop de uitvoer van je programma. Handig als je programma vast loopt.

CHAPTER 41

Git intro and basic use

(in het Engels)

What is Git?

Git is created to have distributed version control. Serious software development is not possible without using some sort of version management. It allows you to get back to a specific version and it enables people to work together on 1 software system.

This Chapter gives first time users a short intro into git. For a lot more info please look at the very good book <https://git-scm.com/book/en/v2> which is free to download.

Create a repository

A repository is a place (typically online) where your code and all previous versions of it are stored. We will now create a repository at the so-called *git-lab* environment at FHICT (only available for FHICT-students; An alternative would be creating a repository at *Github.com*, which is very similar)

Use a browser to go to <https://git.fhict.nl> and create a git repository. Copy the https-url to your clipboard, we need it in a few minutes.

In Git you typically have a repository on a network somewhere where all files of your project are stored, including the history of commits to those files. Furthermore you have a local repository



Figure 41-1 git

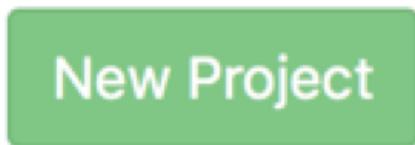


Figure 41-2 new project



Figure 41-3 git https

with all source code versions which you regularly synchronize (using `push` and `pull`) with the remote repository on a server. Other people in your team sync with the same remote repository.

Advanced uses like branching and tagging we will not use in this course.

Clone the repository

Next step is creating a so-called `clone` locally on your laptop. This will be your workspace where you can develop. By keeping it up to date with the server (pushing your changes to the server and pulling the changes of other developers from the server) you can work together with other developers.

You can choose between several tools to do the pulling and pushing, some created to be easy to use and some more advanced. We will use the *command line*: the advantages being that **all** git-functionality is available from the command line and virtually every user using git has this commandline installed. The syntax shown here is that of a Unix-command line (terminal or bash), which is the same for every Unix, Linux or Mac user. If you only have MS-Windows you could install cygwin or adjust the commands a little.

Start a *terminal* or *bash* shell. Go to the directory (*folder*) where you want your workspace to be. How? Suppose you want to create a directory (`mkdir` means *make dir*) `myProject` in your *Documents* directory you type the commands:

```
[ cd Documents
  mkdir myProject
  cd myProject ]
```

`cd` is short for *Change Dir*. Most of the time you don't have to type whole names like *Documents*: just type *Doc* and press the *tab* key and probably the *shell* will complete the name.

In the terminal `clone` the git-repository from the server to your local directory by typing:

```
[ git clone <pasteYourGitUrlHere> ]
```

After that (you will be asked for your *username/password*) you now have a local copy (`clone`) of the entire history of this repository (for a just created project this is still empty of course).

To go inside the local *repository-directory*, which after a `clone` contains the *latest* version of the files, use

```
[ cd <dirname> ]
```

Typing `git status` tells you that (at this moment) there's nothing to commit (which means you don't have made local changes to the repository).

```
Last login: Wed Feb  1 14:19:31 on console
[MacBook-Pro-2:~ fhict$ mkdir sm3
[MacBook-Pro-2:~ fhict$ cd sm3/
[MacBook-Pro-2:sm3 fhict$ git clone https://git.fhict.nl/I889241/Swift.git
Cloning into 'Swift'...
remote: Counting objects: 135, done.
remote: Compressing objects: 100% (111/111), done.
remote: Total 135 (delta 32), reused 0 (delta 0)
Receiving objects: 100% (135/135), 102.60 KiB | 0 bytes/s, done.
Resolving deltas: 100% (32/32), done.
[MacBook-Pro-2:sm3 fhict$ ls
Swift
[MacBook-Pro-2:sm3 fhict$
```

Figure 41-4 bash

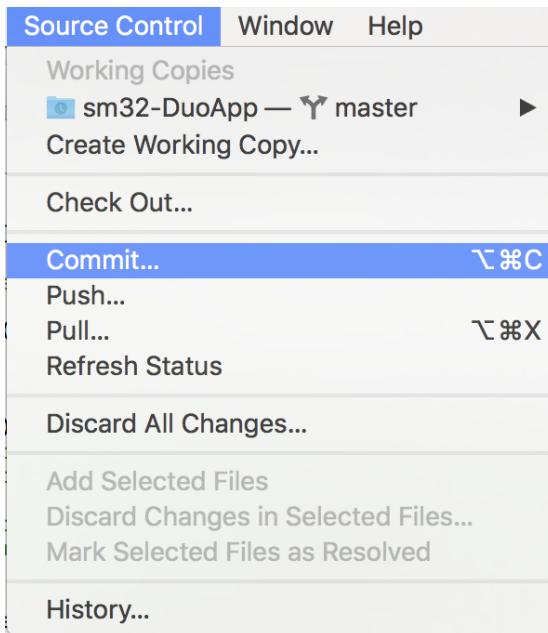


Figure 41-5 git from xcode

These are only the basics of *git*. In my experience most students use it to commit and push their changes right from their IDE. Most *IDEs* and *editors* nowadays have built-in *git*-functionality. Say for example you use Xcode: see xcode-menu ‘Source Control’: when you succeeded with the above doing the same from xcode should be a piece of cake! Good luck!

For more info about *git* please look at the very good book <https://git-scm.com/book/en/v2> which is free to download.



Figure 41-6 git gud

Variabelen

Dit hoofdstuk is geschreven als een *naslagwerk*, het is niet geschikt om uit te leren hoe je met variabelen programmeert.

42.1 Typen variabelen

Een variabele is een stukje geheugen waarin tijdelijk een waarde kan worden opgeslagen. De veelgebruikte typen variabelen zijn:

Inhoud	Naam	Voorbeelden
Stukje tekst	String	"abcde" "dit is een tekst" "" etc.
Geheel getal	Int	12 -1337 0 etc.
Komma getal	Double	10.2 -12.3 5.0 etc.
Waar of niet waar	Bool	true, false

42.2 Variabele aanmaken (declareren)

Variabelen kunnen op verschillende manieren worden aangemaakt, enkele voorbeelden staan hieronder. Merk op dat:

- Je de variabele naam zelf kunt kiezen
- De regel moet worden beeindigd met een ";" -teken

Op verschillende manieren kunnen variabelen worden aangemaakt. Programmeer op een lege regel het type van de variabele (zie hierboven), de naam die je de variabele wil geven (deze kies je zelf) en een ";" teken om het programmeercommando af te sluiten.

Voorbeeld	Effect
String s;	Variabele met de naam s wordt aangemaakt. De default waarde is "".
int i;	Variabele met de naam i wordt aangemaakt. De default waarde is 0.
double d;	Variabele met de naam "d" wordt aangemaakt. De default waarde is 0.0
Bool b;	Variabele met de naam "b" wordt aangemaakt. De default waarde is false
String mijnString;	Variabele met de naam "mijnString" wordt aangemaakt. De default waarde is ""
int getal;	Variabele met de naam "getal" wordt aangemaakt. De default waarde is 0
double straal;	Variabele met de naam "straal" wordt aangemaakt. De default waarde is 0.0

Direct na het aanmaken heeft een variabele een waarde die we de `default waarde` noemen. Dit kan per programmeertaal enigszins verschillen. Daarom is het een goede gewoonte variabelen waarvan je wil dat ze een specifieke waarde hebben deze waarde expliciet toe te kennen.

42.3 Waarde aan variabele geven (toekenning of assignment)

Als een variabele eenmaal is aangemaakt kan hier een waarde aan worden toegekend. Merk op:

- Alleen geldige waarden kunnen worden toegekend (string waarden aan strings, getallen aan int, etc.), het programmeren van een niet geldige toekenning levert een fout op waardoor het programma niet kan worden uitgevoerd.
- De variabele waaraan een waarde moet worden toegekend staat aan de linkerkant van het "=" teken, en de waarde welke in de variabele moet worden gestopt staat rechts van het "=" teken.
- De regel code wordt weer beeindigd met het ";"-teken.

Hier volgen enkele voorbeelden. In commentaar staat erbij uitgelegd wat het betekent.

```

String s;      // maak een variabele aan met naam "s".
s = "test";   // Variabele met de naam "s" krijgt de waarde "test".

int i;
i = 10; // maak variabele met naam "i" aan en geef die waarde 10

double d;
d = 1.52; // Nieuwe variabele genaamd "d" krijgt de waarde 1,52

bool b;
b = true; // Nieuwe variabele "b" krijgt de waarde true

String string1;
string1 = "abc";
String string2;
string2 = string1; // Variabele met de naam "string2" krijgt
                  // de waarde van "string1", namelijk "abc"

int getalA;
getalA = 5;
int getalB;
getalB = getalA; // Variabele met de naam "getalB" krijgt
                  // de waarde van "getalA", namelijk 5

double kommaGetalA;
kommaGetalA = 1.32;

```

```

double kommaGetalB;
kommaGetalB = kommaGetalA; // Variabele met de naam "kommaGetalB" krijgt
                           // de waarde van "kommaGetalA",
                           // namelijk 1.32

String s;
s = textBox1.Text;
// Variabele met de naam "s" krijgt
// als waarde de tekst die in de
// TextBox genaamd "textBox1" staat.

```

Dit werkt omdat de `Text` property van de `TextBox` ook van het type `string` is.

42.4 Variabele aanmaken en direct een waarde geven (declare en initialize)

Variabele met de naam `s` aanmaken en waarde "test" toekennen:

```
[String s = "test";
```

Variabele met de naam `i` aanmaken en waarde 10 toekennen:

```
[int i = 10;
```

Variabele met de naam `d` aanmaken en waarde 1,52 toekennen:

```
[double d = 1.52;
```

Variabele met de naam `b` aanmaken en waarde `true` toekennen:

```
[bool b = true;
```

42.5 Waarden omzetten naar andere typen (convert)

Note Merk op: het omzetten van een `int` of `double` naar een `String` lukt altijd, andersom lukt niet altijd en kan een foutmelding opleveren tijdens het uitvoeren van het programma (`crash` of `Unhandled Exception`).

Note Een `bool` variabele kan niet worden geconverteerd.

Zet de waarde van `i` om naar een tekst met dezelfde waarde. Het resultaat van de laatste regel is dat variabele `s` de waarde 81 krijgt.

```

int i = 81;
String s;
s = Convert.ToString(i);

```

Zet de waarde van `d` om naar een tekst met dezelfde waarde. Het resultaat van de laatste regel is dat variabele `s` de waarde "12.33" krijgt:

```

double d = 12.33;
String s;
s = Convert.ToString(d);

```

Zet de waarde van `s` om naar een geheel getal (`integer`) met dezelfde waarde als dat lukt (anders krijg je een foutmelding). Het resultaat van de laatste regel is dat variabele `i` de waarde 7 krijgt:

```

int i;
String s = "7";
i = Convert.ToInt32(s);

```

Zet de waarde van `s` om naar een `kommagetal` met dezelfde waarde als dat lukt (anders krijg je een foutmelding). Het resultaat van de laatste regel is dat variabele `d` de waarde 12.129 krijgt:

```
[double d;
String s = "12.129";
d = Convert.ToDouble(s);
```

42.6 String bewerkingen (String functies)

Hieronder worden enkele veelgebruikte String functies gedemonstreerd en kort toegelicht.

String's samenvoegen

Met het plus teken kunnen strings aan elkaar worden geplakt.

```
[string tekst = "een tekst.";
string woorden = "Hier staat";
string s = woorden+tekst;
```

De `s` variabele krijgt hier de waarde "Hier staateen tekst." Merk op dat niet automatisch spaties worden toegevoegd.

```
[ string tekst = "tekst.";
  string woorden = "Hier staat";
  string s = woorden + " een " + tekst;
```

Met het "+"-teken kunnen strings aan elkaar worden geplakt. De "`s`" variabele krijgt hier de waarde "Hier staat een tekst."

IndexOf

De plaats van een String binnen een andere String bepalen:

De `Positie` variabele krijgt de waarde 1. Merk op dat de positie van de eerste gevonden "e" in de String wordt gevonden (waarbij vanaf 0 wordt geteld):

```
[string tekst = "regel tekst";
int positie = tekst.IndexOf("e");
```

Er kan ook naar meerdere letters achter elkaar gezocht worden:

```
[string tekst = "regel tekst";
int positie = tekst.IndexOf("tek");
```

De "Positie" variabele krijgt de waarde 6.

Niet gevonden geeft -1:

```
[string tekst = "regel tekst";
int positie = tekst.IndexOf("a");
```

De "Positie" variabele krijgt de waarde -1. De waarde -1 betekent dus: de String komt niet voor binnen de andere String.

Substring

Een stukje uit een string kopiëren:

```
[string tekst = "regel tekst";
string deelTekst = tekst.Substring(0, 1);
```

wat heeft als resultaat dat in `deelTekst` de waarde "r" komt te staan omdat van de oorspronkelijke tekst vanaf positie 0 precies 1 letter gekopieerd wordt.

```
[string tekst = "regel tekst";
 string deelTekst = tekst.Substring(6, 5);
```

Deze code heeft als resultaat dat in deelTekst de waarde "tekst" komt te staan omdat van de oorspronkelijke tekst vanaf positie 6 precies 5 letters gekopieerd worden.

Length

Aantal tekens van de String bepalen. Achter Length hoeven geen haakjes openen en sluiten geplaatst te worden omdat het een property (eigenschap) van de string is en niet een method die je uitvoert.

```
[string tekst = "regel tekst";
 int lengte = tekst.Length;
```

Deze code heeft als resultaat dat in lengte de waarde 11 komt te staan omdat de tekst precies elf lang is. Merk op: dit is inclusief spaties in de tekst. De double quotes om begin en einde van de String waarde aan te geven worden niet meegeteld.

```
[string tekst = "";
 int lengte = tekst.Length;
```

Deze code heeft als resultaat dat in lengte de waarde 0 komt te staan omdat geen tekens in de string staan.

42.7 int en double bewerkingen (operatoren)

Onderstaande bewerkingen zijn zowel op int typen als op double typen van toepassing:

```
[int k;
 k = 5 + 10;
```

Aan variabele k wordt in de laatste regel code de waarde 15 toegekend omdat het +teken de waarden 5 en 10 bij elkaar optelt.

```
[int i = 2;
 int k;
 k = i + 1;
```

Aan variabele k wordt in de laatste regel code de waarde 3 toegekend omdat het +teken de waarden 2 en 1 bij elkaar optelt.

```
[int i = -8;
 int k;
 k = 1 + i;
```

Aan variabele k wordt in de laatste regel code de waarde -7 toegekend omdat het +teken de waarden 1 en -8 bij elkaar optelt.

```
[int i = 5;
 int j = 3;
 int k;
 k = i + j;
```

Aan variabele k wordt in de laatste regel code de waarde 8 toegekend omdat het +teken de waarden uit i en j bij elkaar op telt.

Bij bovenstaande voorbeelden kan de operator (het +teken) worden vervangen door één van de volgende mogelijkheden:

Symbool	Uitwerking
+	Optellen
-	Aftrekken
*	Vermenigvuldigen
/	Delen
%	Geeft de rest na deling. Bijvoorbeeld: $7 \% 5 = 2$ $11 \% 2 = 1$ $6 \% 2 = 0$

Keuzestructuren

Dit hoofdstuk is geschreven als een *naslagwerk*, het is niet geschikt om uit te leren hoe je met keuzestructuren programmeert.

Als een stukje code soms wel en soms niet moet worden uitgevoerd, dan heb je een `if` of `if ... else` statement nodig. Moet een stukje code soms één keer en soms vaker worden herhaald, dan heb je een `for` of `while` statement nodig.

43.1 if-statement

Deze structuur wordt gebruikt om een stukje code uit te voeren afhankelijk van een bepaalde situatie (de conditie genoemd). Algemene vorm:

```
[if ([conditie])
{
    [Uit te voeren code als conditie waar is]
}
```

waarbij conditie is een stelling die de waarde `true` (*waar*) of `false` (*niet waar*) heeft.

Voorbeelden van condities:

Conditie	Betekenis
<code>true</code>	Waar
<code>false</code>	Niet waar
<code>i > 5</code>	Is i groter dan 5?
<code>i < 7</code>	Is i kleiner dan 7?
<code>i >= 1</code>	Is i groter of gelijk aan 1?
<code>i <= 2</code>	Is i kleiner of gelijk aan 2?
<code>i == 3</code>	Is i precies gelijk aan 3?
<code>i != 3</code>	Is i ongelijk aan 3?
<code>stukjeText == "abcde"</code>	Is <code>stukjeText</code> precies gelijk aan "abcde"?
<code>stukjeText < "abcde"</code>	Komt <code>stukjeText</code> eerder in het alfabet dan "abcde"?
etc.	

Verder staat *[Uit te voeren code als conditie waar is]* voor een stukje code (dit kunnen meerdere regels code zijn) dat moet worden uitgevoerd als de conditie `true` (*waar*) is.

Als precies één regel code moet worden uitgevoerd zou je ervoor kunnen kiezen de accolades openen en sluiten weg te laten, maar dit maakt de kans op bugs een stuk groter, dus dat raden we af.

43.2 if ... else ... statement

Een if statement kan uitgebreid worden met een "else" blok. Als de conditie niet "waar" oplevert dan wordt de code in het else blok uitgevoerd. Algemene vorm:

```
if ([conditie])
{
    [Uit te voeren code als conditie waar is]
}
else
{
    [Uit te voeren code als conditie niet waar is]
}
```

Merk op: of de conditie nu wel of niet waar is, altijd wordt één van de twee stukjes code uitgevoerd.

43.3 Voorbeelden "if ..." statement en "if ... else ..." statement

```
if (true)
{
    TextBox1.Text = "test";
}
```

Het stukje code tussen { en } wordt altijd uitgevoerd, dus de Text van de *TextBox* wordt altijd "test" gemaakt.

```
if (false)
{
    TextBox1.Text = "test";
}
```

Het stukje code tussen { en } wordt nooit uitgevoerd.

```
bool b = true;
if (b)
{
    TextBox1.Text = "test";
}
```

Als b de waarde *true* (= *waar*) heeft wordt de *Text* in de *TextBox* "test" gemaakt. Dit is hier nu altijd het geval omdat in dit stukje code aan variabele b alleen de waarde "true" wordt toegekend.

```
int i = 10;
if (i < 5)
{
    i = i + 1;
}
```

Als getal i kleiner dan 5 is, dan wordt bij de waarde van i één opgeteld, anders gebeurt er niets.

```
TextBox1.Text = "test2";
if (TextBox1.Text != "test")
{
    TextBox1.Text = "test3";
}
```

Als de tekst in de textbox niet gelijk is aan "test" (dat is hier het geval) dan wordt de tekst van de textbox veranderd in "test3".

```
if (true)
{
    TextBox1.Text = "test";
```

43.3 Voorbeelden "if ..." statement en "if ... else ..." statement

```
    }
} else
{
    TextBox1.Text = "test2";
}
```

Het stukje code tussen de eerste { en } wordt altijd uitgevoerd, dus de Text van de TextBox wordt altijd "test" gemaakt. Het stukje code tussen de tweede { en } wordt nooit uitgevoerd.

```
int i = 5;
if (i >= 10)
{
    i = i + 1;
}
else
{
    i = i + 5;
}
```

Als getal i groter of gelijk aan 10 is dan wordt bij getal i 1 opgeteld. Dit is hier niet het geval, dus wordt bij i 5 opgeteld. Resultaat: i krijgt de waarde 10.

```
int i = 5;
if (i >= 10)
{
    i = i + 1;
}
else
{
    i = i + 5;
    if (i >= 10)
    {
        i = 20;
    }
}
```

Als getal i groter of gelijk aan 10 is dan wordt bij getal i 1 opgeteld. Dit is hier niet het geval, dus wordt bij i 5 opgeteld. Resultaat: i krijgt de waarde 10, vervolgens wordt gecontroleerd of $i \geq 10$, dat is nu het geval dus krijgt i uiteindelijk de waarde 20 toegekend.

CHAPTER 44

Herhalingsstructuren

Dit hoofdstuk is geschreven als een *naslagwerk*, het is niet geschikt om uit te leren hoe je met herhalingsstructuren programmeert.

44.1 while statement

Deze structuur wordt gebruikt om een stukje code uit te voeren zolang aan bepaalde voorwaarden is voldaan. Dit varieert van 0 keer de code uitvoeren tot het in de oneindigheid aantal keer uitvoeren van de code).

Algemene vorm:

```
[ while ([conditie])
  {
    [Uit te voeren code zolang de conditie waar is]
  }
```

Note Na de eerste regel staat geen ";" teken.

Note Eerst wordt gecontroleerd of aan een voorwaarde is voldaan, dan pas wordt eventueel code uitgevoerd.

44.2 do while statement

Deze structuur wordt gebruikt om een stukje code uit te voeren. Elke keer nadat het stukje code is uitgevoerd wordt gecontroleerd of nog aan bepaalde voorwaarden is voldaan, zo ja, dan wordt de code opnieuw uitgevoerd. Het aantal keer uitvoeren van de code varieert van 1 keer de code uitvoeren tot het in de oneindigheid aantal keer uitvoeren van de code.

Algemene vorm:

```
[ do
  {
    [Uit te voeren code zolang de conditie waar is]
  } while ([conditie]);
```

Note Na de laatste regel staat een ";" teken.

Note Eerst wordt de code één keer uitgevoerd, dan pas wordt gecontroleerd of de code eventueel vaker moet worden uitgevoerd.

44.3 Voorbeelden while en do while statement

```
int i = 0;
while(i < 10)
{
    MessageBox.Show("Test");
    i = i + 1;
}
```

Variabele *i* krijgt in het begin de waarde 0 en er wordt net zo lang doorgegaan met *MessageBoxes* weergeven totdat *i* kleiner dan 10 is. De code wordt dus doorlopen met achtereenvolgens de waarden 0, 1, 2, 3, 4, 5, 6, 7, 8 en 9. Er worden daarom 10 *Messageboxes* getoond met de tekst "Test".

```
int i = 5;
while(i > 0)
{
    MessageBox.Show("Test");
    i = i - 1;
}
```

Variabele *i* krijgt in het begin de waarde 5 en er wordt direct gestopt als *i* de waarde 0 krijgt toegekend. De code wordt dus doorlopen met de waarden 5, 4, 3, 2, 1. Er worden daarom 5 *Messageboxes* getoond met de tekst "Test".

```
int i = 10;
do
{
    MessageBox.Show("Test");
    i = i + 1;
} while (i < 5);
```

Variabele *i* krijgt in het begin de waarde 10, de code wordt uitgevoerd, en vervolgens wordt net zo lang doorgegaan met *Messageboxes* weergeven totdat *i* kleiner dan 5 is. De code wordt dus doorlopen met de waarde 10. Er wordt daarom 1 *Messagebox* getoond met de tekst "Test".

44.4 for statement

Deze structuur wordt gebruikt om een stukje code een vooraf bekend aantal keer uit te laten voeren.

Algemene vorm:

```
for([teller variabele aanmaken]; [herhalingsconditie]; [teller variabele aanpassen])
{
    [herhaaldelijk uit te voeren code]
}
```

waarbij *[teller variabele aanmaken]* een variabele met zelf te kiezen variabelenaam wordt aangeemaakt en van een waarde voorzien. Veel gebruikte variabele namen voor een for statement zijn "i", "j", "k" omdat deze een hele korte naam hebben, dat leest in veel gevallen prettig. Ook "index", "count" of "teller" worden vaak gebruikt. Het type variabele is meestal int. De waarde waarmee de teller wordt gevuld is afhankelijk van wat je aan het programmeren bent. In veel gevallen heeft deze de waarde 0.

Voorbeelden:

```
[ int i = 0
[ int j = 100
```

Dan [herhalingsconditie]: deze uit te voeren code wordt net zo lang herhaald als uit de voorwaarde de waarde true komt. Hierin verwijst je naar de *teller* variabele.

Voorbeelden:

```
[ i < 10
[ j > 0
```

[teller variabele aanpassen] Het verhogen of verlagen van de teller. Vaak wordt deze met 1 verhoogd of verlaagd, soms in grotere stappen (bijv. 10).

Voorbeelden:

```
[ i = i + 1
[ j = j - 10
```

[herhaalbaar uit te voeren code] Het stukje code (dit kunnen meerdere regels code zijn) dat moeten worden uitgevoerd zolang de herhalingsconditie "true" (waar) is.

Ieder forstatement is om te zetten naar een while statement dat hetzelfde doet, en andersom.

44.5 Voorbeelden for statement

```
[ for(int i =0 ; i < 10 ; i = i + 1)
{
    MessageBox.Show("Test");
}
```

Variabele i krijgt in het begin de waarde 0 en er wordt direct gestopt als i de waarde 10 krijgt toegekend. De code wordt dus doorlopen met de waarden 0,1,2,3,4,5,6,7,8 en 9. Er worden daarom 10 messagebox-en getoond met de tekst "Test".

```
[ for(int i =5;i > 0; i = i - 1)
{
    MessageBox.Show("Test");
}
```

Variabele i krijgt in het begin de waarde 5 en er wordt direct gestopt als i de waarde 0 krijgt toegekend. De code wordt dus doorlopen met de waarden 5,4,3,2,1. Er worden daarom 5 messagebox-en getoond met de tekst "Test".

```
[ for(int i =0;i < 10;i++)
{
    MessageBox.Show("Test");
}
```

Hetzelfde resultaat als het eerste voorbeeld, maar dan in een verkorte schrijfwijze:

```
[ i = i + 1;
```

is hetzelfde als

```
[ i++;
```

Hetzelfde resultaat als het tweede voorbeeld, maar dan in een verkorte schrijf-wijze:

```
[ for(int i =5;i > 0; i--)
{
    MessageBox.Show("Test");
}
```

```
[ i=i-1;
```

is hetzelfde als

```
[ i--;
```

De code

```
[ for(int i =0;i < 10; i++)
{
    MessageBox.Show("Test "+i);
}
```

heeft als resultaat dat *MessageBoxes* verschijnen met achtereenvolgens:

```
"Test 0"
"Test 1"
"Test 2"
"Test 3"
"Test 4"
"Test 5"
"Test 6"
"Test 7"
"Test 8"
"Test 9"
```

De code

```
[ for(int i =5;i > 0; i = i - 2)
{
    MessageBox.Show("Test "+i);
}
```

laat messagebox-en verschijnen met achtereenvolgens:

```
"Test 5"
"Test 3"
"Test 1"
```

en tot slot geeft

```
[ for(int y =0;y < 2; y++)
{
    for(int x =0;x < 3; x++)
    {
        MessageBox.Show("(" +x+ "," +y+ ")");
    }
}
```

als resultaat *MessageBoxes* verschijnen met:

```
"(0,0)"
"(1,0)"
"(2,0)"
"(0,1)"
"(1,1)"
"(2,1)"
```

Methoden

Dit hoofdstuk is geschreven als een *naslagwerk*, het is niet geschikt om uit te leren hoe je met methodes programmeert.

45.1 Algemene structuur methoden

Een methode is een stukje code dat vanuit een ander stukje code is aan te roepen. Als een methode een waarde terug geeft welke gebruikt gaat worden in het stukje code waarvanuit deze is aangeroepen spreek je over een methode welke "een waarde teruggeeft". Ook kunnen aan een methode één of meer waarden worden meegegeven. Dit worden argumenten genoemd.

45.2 Belangrijkste voordelen van het gebruik van methoden:

1. Overzichtelijkheid: Als alle code in één enkele event handler (bijv. *ButtonX_Click*) wordt geplaatst wordt je code al snel erg overzichtelijk.
2. Werk verdelen: Als je voordat je gaat programmeren het programmeerwerk wilt verdelen kun je de te maken code opdelen in methoden en deze met verschillende programmeurs tegelijkertijd programmeren.
3. Onderhoudbaarheid & herbruikbaarheid: Als je op verschillende plaatsen in je programma hetzelfde stuk code vaker uit wilt voeren kun je vanaf de verschillende plaatsen een methode aanroepen die je maar één keer hoeft te programmeren. Dat scheelt code en is gemakkelijker te onderhouden dan dat je code verschillende keren in je programma kopiert en plakt.

Een methode heeft de volgende structuur:

```
[private [returnType] [methodeNaam]([parameters])  
{  
    ...  
    [return returnWaarde]  
}
```

`private` geeft aan dat de methode alleen binnen het huidige bestand (lees: *Form1*) kan worden aangeroepen. Wat dit precies inhoudt is voor dit vak niet interessant, hier wordt later op teruggekomen.

`[returnType]` Het type van de waarde die de methode terug geeft. Als de methode geen waarde terug geeft, is dit `void` (*niets*). Voorbeelden: `int`, `double`, `bool`, `string`, `void`.

`[methodeNaam]` Zelf gekozen naam voor de methode, te vergelijken met een zelf gekozen naam voor een variabele. Voorbeelden: `TelOp`, `ToonMelding`, `Methode1`, `Abc`.

[parameters] Optioneel onderdeel. Hiermee wordt opgegeven welk(e) type(n) waarde(n) moet worden meegegeven aan de methode en onder welke naam deze waarde binnen de method kunnen worden gebruikt. Meerdere parameters worden gescheiden met een ","-teken. Voorbeelden: int deler, int getalA, int getalB, bool isIngelogd, double eenKommaGetal.

[return returnWaarde] Met return" gevuld door een waarde die aan het [returnType] voldoet wordt een waarde teruggegeven vanuit de methode aan het stukje code dat de methode heeft aangeroepen. Als [returnType] void is hoeft er geen return worden gebruikt. Voorbeelden: return uitkomst;, return 10;, return mijnTekst;, return "Hallo"+" daar!";, return getal > 10;

45.3 Voorbeelden Methoden

Een aantal voorbeeldmethoden:

```
private int AddTwoNumbers(int number1, int number2)
{
    int som;
    som = number1 + number2
    return som;
}

private int SquareANumber(int number)
{
    return number * number;
}
```

Bovenstaande methoden zijn als volgt aan te roepen:

```
[ int sum = AddTwoNumbers(8765, 287);
of:
[ int kwadraat = SquareANumber(63);
of, beiden:
[ int total = SquareANumber(AddTwoNumbers(1, 2));
```

Externe tekst en uitleg (engelstalig) over methoden en parameters

Zie bijvoorbeeld C-sharpcorner over methods¹ voor meer uitleg.

¹http://www.c-sharpcorner.com/UploadFile/myoussef/CSharpMethodsP_111152005003025AM/CSharpMethodsP_1.aspx