OIS12

FHICT software docenten, met name Inge van Engeland, Marcel Veldhuijzen, Jan Oonk,

January 15, 2018
master @ 94825f9*

Layout and typography based on the sbabook LaTeX class by Damien Pollet.

Contents

IV ProgrammeerConcepten

V Challenges

Illustrations

Part I

Course OlS12

Dit document is nog in ontwikkeling: het kan zijn dat hier of daar nog wat ontbreekt! De volledige inhoud van het vak staat ook op Canvas.

CHAPTER 1

OIS12 Home

1.1 Wat wil jij programmeren?

OIS12 bereidt je voor op semester 2 van ICT en Technology en ICT en Software Engineering. Aan het eind van OIS12 kun je bruikbare onderhoudbare programma's schrijven in C#.

1.2 Boek

We gebruiken geen boek bij OIS12. Als je het fijn vindt om toch met behulp van een boek te studeren dan raden we het boek Handboek Visual C# 2012 aan door Dirk Louis. Dit boek is tevens handig voor het vak FUN12.

Een ander goed boek waarin je goed de leerdoelen van OIS12 kunt vinden en kunt leren a.d.h.v. theorie en opdrachten is: Programmeren in C# door Douglas Bell en Mike Parr (Nederlandstalig) van uitgeverij Pearson.

1.3 Leeractiviteiten

Je krijgt periodiek feedback van je docent die wordt genoteerd in Canvas. Bekijk zelf aan de hand van deze feedback welke programmeer-oefeningen je gaat maken. Bij OIS12 ga je zo veel mogelijk zelf aan de slag met programmeren. Door veel te oefenen krijg je het vak snel onder de knie.

1.4 Beoordeling

Je eindcijfer wordt bepaald door de teksten van de feedback die je gedurende de lesweken hebt ontvangen. Alle leerdoelen aangetoond? Dan heb je zeker een voldoende voor OIS12. Wil je graag een 9 of 10 halen voor OIS12? Maak een afspraak met je docent hoe je dat kunt aanpakken.

1.5 Blokboek/studentenhandleiding

Zie intranet-portal

Wij wensen je een fijne en leerzame cursus.

CHAPTER 2

Leerdoelen OIS12

Deze leerdoelen komen uit de vakkengids voor het startsemester. Globale doel is dat je alle onderstaande leerdoelen afzonderlijk en gezamenlijk kunt toepassen in een C#-programma dat je schrijft aan de hand van een functionele specificatie. Let op: je moet dit alles uit jezelf kunnen programmeren, alleen code kunnen uitleggen is niet voldoende!

2.1 Leerdoelen VOORSTEL OIS12 cohort201802

Dit zijn de minimale leerdoelen om tot een voldoende indicatie te komen voor het vakgebied.

We beginnen met de centrale technische leerdoelen van OIS12.

constructor

Je kunt in C# vanuit een specificatie constructors programmeren en deze gebruiken.

Bij FUN12 krijg je geleerd hoe je met `classes` moet werken, OIS12 voegt hier aan toe dat je `constructors` gebruikt om `instanties` van een `class` te maken.

private/public

Je weet wat `private` en `public` betekenen en maakt `fields` in een `class` altijd `private`.

Als van buiten een `object` de waarde van een `field` moet wijzigen gaat dit middels hiertoe bestemde `methods` of `properties`. Dit is niet iets wat je 1

keer laat zien maar wat je vanaf dat moment in AL je programma's doet! (dus ook bij FUN12!)

overloading

Je kunt methodes programmeren met dezelfde naam binnen 1 class en je kunt uitleggen wat method overloading betekent.

We gebruiken dit bij OIS12 veel voor constructors (waarbij dit ook kan, net als bij 'gewone' methodes).

property

Je snapt wat een `property` is (zowel *normaal* als *auto property*) en hebt laten zien dat je ze zelf kunt aanmaken.

Het snappen wat een property is is het belangrijkste aangezien veel voorbeelden op internet met `properties` werken: Mensen die bijvoorbeeld al `javakennis` hebben en (ná aantonen dat ze snappen wat een `property` is) voorkeur hebben voor `Get` en `Set methods` mogen dat ook.

List en foreach

Je kunt programmeren met foreach en lijsten van objecten.

Met een lijst wordt hier bedoeld een `List<T>` waarbij T een geldig `type` is, bijvoorbeeld een zelf geprogrammeerde `class`.

Van relation tussen classes naar een List

Je kunt vanuit een gegeven klassendiagram relaties in `C#` programmeren waarbij je gebruik maakt van Lists (lijsten van objecten).

Als een `object` meerdere `objecten` van een `type` `T` bevat wordt dit gerealiseerd door eerstgenoemd `object` een `Field` te geven van het type `List<T>`.

enum

Je kunt `enums` programmeren.

Dit wil zeggen dat je weet hoe je dit doet, een keer een goed voorbeeld hebt laten zien aan de docoent. Verder hou je je ogen open tijdens het ontwikkelen: *als* je een situatie tegen komt die roept om een `enum` gebruik je die. Enums kunnen een belangrijk wapen zijn om je code leesbaar en onderhoudbaar te houden.

Nu een aantal leerdoelen waarbij je mag laten zien dat je een professional in wording bent.

Professionele houding

Je stelt je professioneel op. Dit blijkt ondermeer uit je aanwezigheid, actieve deelname tijdens de lessen, opbouwend kritische houding, samenwerken met anderen en het houden aan afspraken.

Als je er een keer tijdens lestijden niet bent, pas later aanwezig kunt zijn of eerder weg gaat communiceer je dit naar de docent. Ook als de les langer duurt dan jij zonder roken, gamen of facebook kunt en je dus naar buiten gaat (roken) of naar een open ruimte (gamen/facebook): dat meld je even bij de docent.

Visual Studio

Je kunt je eigen Visual Studio-installatie onderhouden.

Ook dit leerdoel gaat in belangrijke mate over professionaliteit: Zorg dat je je laptop in orde hebt, met visual studio werkend. Natuurlijk kunnen er wel eens problemen zijn, maar je zorgt dat je een backup hebt zodat je geen weken werk kwijt bent. Als je laptop of Visual Studio niet orde zijn weet je binnen afzienbare tijd te regelen dat je toch aan het vak kunt werken.

Analyse

Je kunt een analysedocument schrijven waarbij je voorafgaand aan een project een gepriotiriseerde opsomming geeft van functionele eisen aan een applicatie.

Natuurlijk kan van een startemesterstudent nog geen volledig analysedocument verwacht worden. Wat we echter wel verwachten is dat je requirements verwoordt en hier een MoSCoW-indeling van maakt op een manier die laat zien dat je de klant serieus neemt. Dat wil zeggen dat je aandacht en liefde erin stopt: Verzorgde layout en geen taalfouten. Tot slot verwerk je eventuele feedback op documenten.

Technical design

Je bent in staat om vooraf te ontwerpen uit welke klassen je applicatie bestaat, en wat de eigenschappen, methoden en verantwoordelijkheden van die klassen zijn, de relaties tussen deze classes en dit alles op een grafische manier weer te geven.

Het gaat er niet om dat je (bijvoorbeeld) `UML` ('Unified Modeling Language') volgens alle regels kunt toepassen: het gaat er om dat je voordat je gaat coderen op een (enigszins abstracte) manier over een programma kunt praten en hier een grafische weergave kunt maken. Je zult ook hier mogelijk feedback op krijgen waar je iets mee zult moeten doen.

UI separation

Je bent in staat om kleine applicaties te programmeren waarbij er een goede scheiding is tussen code in klassen en userinterface-code.

Met de kennis en vaardigheden die je nu hebt is deze scheiding nog niet altijd mogelijk maar we streven dit zoveel als mogelijk wel na.

De nu volgende leerdoelen zitten in dit vak omdat je ze logischerwijs bij een ander leerdoel tegenkomt (collateral).

FileHandling

Je moet vanuit een specificatie een C#-programma kunnen schrijven waarmee tekstbestanden kunnen worden ingelezen en worden geschreven.

Exception

Je kunt op een correcte manier excepties in je C#-programma afhandelen.

Wat exceptions betreft gaat hier om het gebruik van de `try..catch` op momenten dat dit van toepassing is. Net zo belangrijk is dat je het *alleen dan* gebruikt!

graphics

Je kunt vanuit een specificatie een programma schrijven dat werkt met de diverse functies van de Windows GDI (graphics).

static

Je kunt gebruik maken van bestaande methoden die static zijn en je kunt uitleggen wat het static keyword betekent.

casting

Je weet wat een `cast` is en kunt `casting` toepassen in het geval een `GUI-control` verwijst naar een of meerdere `objecten` van een bepaald type.

Hét voorbeeld hiervan binnen OIS12 is dat je een `object` uit een `ListBox` haalt waarvan je zeker weet dat het van een specifieke `class` als `type` heeft: Als je een element uit de `Items` haalt kun je de waarde `casten` naar het juiste `type`.

Verder weet je dat `cast` inherent *onveilig* is (zeker in sommige programmeertalen) en dus niet onnodig gedaan moet worden.

Value versus Reference Types

Je kunt het verschil tussen value en reference types uitleggen en er mee programmeren.

Dit is niet een leerdoel dat je expliciet moet aantonen.

XAML

Je kunt user interfaces voor applicaties maken met zowel de Windows Forms designer als met de XAML designer.

2.2 Leerdoelen OIS12 cohort201708

visual studio

Je kunt je eigen Visual Studio-installatie onderhouden. In feite gaat dit leerdoel over professionaliteit: Zorg dat je je laptop in orde hebt, met visual studio draaiend. Natuurlijk kunnen er wel eens problemen zijn, maar je zorgt dat je een backup hebt zodat je geen weken werk kwijt bent. Als je laptop of visual studio niet orde zijn weet je binnen korte tijd te regelen dat je toch aan het vak kunt werken.

analyse

Je kunt een analysedocument schrijven waarbij je voorafgaand aan een project een gepriotiriseerde opsomming geeft van functionele eisen aan een applicatie.

filehandling

Je moet vanuit een specificatie een C#-programma kunnen schrijven waarmee tekstbestanden kunnen worden ingelezen en worden geschreven.

exception

Je moet op een correcte manier excepties in je C#-programma kunnen afhandelen.

UML design

Je bent in staat om vooraf te documenteren uit welke klassen je applicatie bestaat, en wat de eigenschappen, methoden en verantwoordelijkheden van die klassen zijn en deze grafisch in UML weer te geven.

associaties

Je kunt associaties (relaties) met multipliciteiten tussen klassen ontwerpen en deze weergeven in een UML-klassendiagram.

constructor

Je kunt in C# vanuit een specificatie constructoren programmeren en deze gebruiken.

overloading

Je kunt in C# binnen een klasse methoden programmeren met dezelfde naam en je kunt uitleggen wat method overloading betekent.

private/public

Je kunt programmeren met private en public.

property

Je kunt property's programmeren vanuit een gegeven specificatie.

enum

Je kunt enum's programmeren.

graphics

Je kunt vanuit een specificatie een programma schrijven dat werkt met de diverse functies van de Windows GDI (graphics).

static

Je kunt gebruik maken van bestaande methoden die static zijn en je kunt uitleggen wat het static keyword betekent.

casting

Je kunt programmeren met casting in de context van het casten van een object naar het gewenste subtype.

List en foreach

Je kunt programmeren met foreach en lijsten van objecten.

professionele houding

Je stelt je professioneel op. Dit blijkt ondermeer uit je aanwezigheid, actieve deelname tijdens de lessen, een kritische houding en samenwerken met anderen.

Van UML relation naar List

Je kunt vanuit een gegeven UML-klassendiagram relaties in C# programmeren waarbij je gebruik maakt van List's (lijsten van objecten).

Value versus Reference Types

Je kunt het verschil tussen value en reference types uitleggen en er mee programmeren.

UI separation

Je bent in staat om kleine applicaties te programmeren waarbij er een goede scheiding is tussen code in klassen en userinterface-code.

XAML

Je kunt user interfaces voor applicaties maken met zowel de Windows Forms designer als met de XAML designer.

2.3 Proftaak PT12

Vanuit de proftaak PT12 vragen de studenten twee maal (1x rond week 3 en 1x rond week 7) mondeling feedback aan de OIS12-vakdocent (het initiatief ligt bij de studenten). Die feedback wordt door de docent mondeling gegeven, de studenten noteren die feedback en leveren die in in hun PT12-course. Bij de afronding van PT12 wordt ook beoordeeld wat de studenten met die feedback hebben gedaan.

Als docent wil je graag een `class diagram` zien van het project. Let hierbij op dat het ook daadwerkelijk een `class diagram` (UML-ish notatie) is en geen `database model`. Bespreek het model met het team. Na dit gesprek kan de docent kijken of het leerdoel over het modelleren aangetoond is (wellicht 'once'?).

2.4 De relatie met FUN12

Deze paragraaf gaat over de relatie met FUN12 en dus wat bekend wordt verondersteld bij OIS12.

FUN12 is ontstaan doordat een deel van het oude vak SE12 (grofweg programmeren met `classes`, `constructors`, `objecten`) is samengevoegd met een deel van het oude vak DBS12 over de beginselen van `relationele databases` en `SQL`. De rest van SE12 is hernoemd naar OIS12, hoewel deze rest niet zonder `classes`, `constructors` en `objecten` kan). Vandaar dat er een overlap zit tussen OIS12 en FUN12. Aangezien deze onderwerpen bij het maken van Software zo belangrijk zijn is dit ook helemaal niet erg.

Part II

Course: PAC, Pre Assessment Course

2.5 Wat is de Pre Assessment Course?

De Pre Assessment Course is een zelfstudiecursus die studenten doen in het kader van de MVV.

Een goed boek waaruit je kunt leren programmeren en kunt leren aan de hand van theorie en opdrachten is: **Programmeren in C#** door *Douglas Bell en Mike Parr* (Nederlandstalig) van uitgeverij Pearson.

2.6 Leeractiviteiten

Dit is een zelfstudiecursus. De student leert programmeren en oefent hiermee.

2.7 Beoordeling

Aan het eind vindt er een assessment plaats waarin de student het gemaakte werk toont en de assessoren bepalen of de student het verkorte traject in mag.

2.8 Blokboek/studentenhandleiding

Zie intranet-portal

2.9 Leerdoelen Pre Assessment S-profiel voor master TU/e

> **To do** Verdeling van leerdoelen over Coen (C) en Nico (N): C: 10 t/m 18; N: 4 t/m 9 en 19 t/m 22

Oriëntatie op de ICT-context

1. De student kan uitleggen wat voor soort werkzaamheden een HBO ICT bachelor - afhankelijk van het verkozen profiel - uitvoert. Een profiel is een combinatie van ICT met één van de volgende vier invalshoeken: Business (B), Media Design (M), Software Engineering (S) en Technology (T).

2. De student kan motiveren waarom het S-profiel de voorkeur heeft.

3. De student kan uitleggen wat voor soort werkzaamheden de software engineer uitvoert tijdens analyse, ontwerp, realisatie, testen en beheer.

Basis programmeren

4. De student kan progammacode schrijven waarbij er aandacht is voor onderhoudbaarheid in de vorm van: documentatie van programmacode, het kiezen van betekenisvolle namen van variabelen/methoden en het voldoen aan afgesproken naamsconventies.

5. De student kan keuzestructuren (if), herhaalstructuren (for, while) en methoden programmeren.

6. De student kan gebruikmaken van een debugger met breakpoints, inspectievensters, 'step into' en 'step over'.

7. De student kan eigenschappen van GUI-objecten zowel visueel als in programmacode aanpassen.

8. De student kan zelfstandig zoeken naar nuttige eigenschappen/events van GUI-objecten.

Programmeren met objecten

9. De student weet wat de scope van een variabele is.

10. De student kan gebruikmaken van bestaande methoden die static zijn en kan uitleggen wat het static keyword betekent.

11. De student kan collecties, en in het bijzonder arrays en lijsten, toepassen.

12. De student kan een foreach-herhaalstructuur (C#) voor een collectie programmeren.

13. De student kan gegevens naar een tekstbestand schrijven en gegevens vanuit een tekstbestand inlezen.

14. De student kan excepties in programmacode afhandelen.

Ontwerp van klassen

15. De student kan uitleggen wat klassen zijn en kan een zelf gedefinieerde klasse met constructor(en), eigenschappen (C#) en methoden programmeren.

16. De student kan associaties (met multipliciteit) tussen klassen ontwerpen, in de vorm van een UML-klassendiagram.

17. De student weet wanneer de private en public keywords moeten worden gebruikt.

18. De student kan kleine applicaties programmeren waarbij er een goede scheiding is tussen programmacode in zelf gedefinieerde klassen en programmacode ten behoeve van de GUI.

Structured Query Language (SQL)

19. De student kan informatie opvragen uit een relationele database. De student kan hierbij de volgende SQL-concepten toepassen: (sub)query, select, insert, delete, inner join, group by en standaardfuncties.

- Eenvoudige query met sleutelwoorden SELECT, FROM, WHERE en toepassing van de sleutelwoorden AND, OR, NULL, BETWEEN, IN EN LIKE.
- Toepassing van de 5 basisfuncties MIN, MAX, COUNT, AVG, SUM.
- Toepassing van subqueries.
- Toepassing van inner join.
- Toepassing van group by.

Databases

20. De student kan een Entity Relationship Diagram (ERD) ontwerpen met ten minste drie gerelateerde entiteiten.

21. De student kan vanuit een gegeven ERD, inclusief een veel-op-veel relatie, een relationele database met primary en foreign keys definiëren.

22. De student kan met behulp van de Structured Query Language (SQL) informatie toevoegen aan (INSERT) en verwijderen uit (DELETE) een relationele database.

23. De student kan een relationele database integreren binnen een applicatie.

Part III

Algemene info

CHAPTER 3

CodingGuidelines

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 onderhoudbaardere code oplevert als je er vanaf wijkt. Anderzijds is het sowieso eerst de *Reeglen 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.

CHAPTER 4

External Resources

Enkele alternatieve bronnen:

Het youtube-kanaal van FHICT-docent Wilrik

YouTube Wilrik[1]

Brackeys

Brackeys[2]

The New Boston

Je kunt ook kijken naar De video-tutorials van The New Boston[3]

Hier de indeling:

[1] https://www.youtube.com/watch?v=IuStUzWuPrw
[2] https://www.youtube.com/watch?v=pSiIHe2uZ2w&list=PLPV2KyIb3jR6ZkG8gZwJYSjnXxmfPAl51
[3] 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

Part IV

ProgrammeerConcepten

CHAPTER 5

Herhaling OIS11

5.1 Werken met variabelen in C#

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

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.

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;
string 1 = "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.

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;
```

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);
```

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.

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

5.2 Werken met keuzestructuren in C#

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.

`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
true	Waar
false	Niet waar
i > 5	Is i groter dan 5?
i < 7	Is i kleiner dan 7?
i >= 1	Is i groter of gelijk aan 1?
i <= 2	Is i kleiner of gelijk aan 2?
i == 3	Is i precies gelijk aan 3?
i != 3	Is i ongelijk aan 3?
stukjeText == ”abcde”	Is *stukjeText* precies gelijk aan ”abcde”?
stukjeText < ”abcde”	Komt *stukjeText* eerder in het alphabet 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.

`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.

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";
}
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 >= 10`, dat is nu het geval dus krijgt *i* uiteindelijk de waarde `20` toegekend.

`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.

`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.

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"`.

`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 aangemaakt 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 verwijs 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
```

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

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

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 schrijfwijze:

```
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)"
```

5.3 Werken met methoden in C#

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 waar vanuit 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.

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`” gevolgd 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;`

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));
```

Tekst en uitleg (engelstalig) over methoden en parameters

Zie bijvoorbeeld C-sharpcorner over methods[1] voor meer uitleg.

5.4 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.

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

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 (`play`knop).
Shift-F5		Stop de uitvoer van je programma. Handig als je programma vast loopt.

5.5 Online C-sharp tutorials

MSDN tutorial

MSDN How do I Learn C# tutorials (Engelstalig)[2]

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)[3]

Bruikbaarheid: ****

[2] http://msdn.microsoft.com/en-us/vcsharp/aa336766.aspx
[3] http://www.techzine.nl/tutorials/358/3/c-les-1-beginnen-met-c-de-eerste-stapjes.html

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)[4]

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.

Blackwasp tutorial

BlackWasp[5]

Bruikbaarheid: ***

Toelichting: Verzameling tutorials en artikelen (Engelstalig).

5.6 Online C-sharp boeken

C# Station Tutorial (Engelstalig)[6] 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)[7] 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.

[4] http://www.sitemasters.be/tutorials/17/1/564/CSharp.NET/CSharp_Zelf_een_WebBrowser_maken
[5] http://www.blackwasp.co.uk/
[6] http://csharp-station.com/
[7] http://www.robmiles.com/c-yellow-book/Rob%20Miles%20CSharp%20Yellow%20Book%202010.pdf

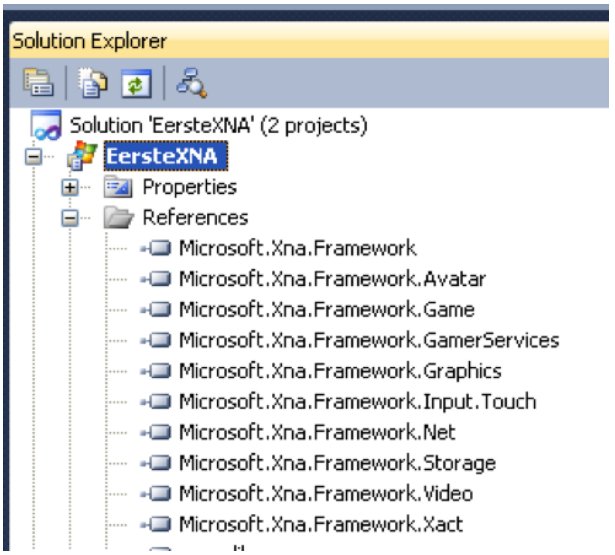

Figure 5-1 eerste XNA

5.7 Bruikbare technieken proftaak

3d objecten

In de eerste opdracht over XNA heb je met plaatjes gewerkt en niet met 3D-modellen. 3D is mooier, echter en interessanter. Maar ook lastiger te implementeren.

Op Internet zijn aardige voorbeelden te vinden.

Probeer eerst deze uit: MSDN over XNA en 3D[8]

WIIMote

In de opdracht ”1e XNA” heb je kunnen zien dat het XNA framework was toegevoegd aan Visual Studio 2010. Dit is te vinden in de solution Explorer onder Reference.

In het programma gebruikte je het keyword `using` om gebruik te maken van de mogelijkheden van het Framework.

Om een component te gebruiken in Visual Studio ga je altijd op deze manier te werk. Het XNA framework heb je geïnstalleerd en daarbij werden de referentie aan VS toegevoegd. Net zoals het XNA framework aan Visual Studio 2010 is toegevoegd kun je een softwarecomponent toevoegen om de Wiimote te gebruiken. Deze component is te vinden op het Internet genaamd

[8] http://msdn.microsoft.com/en-us/library/bb203897(v=xnagamestudio.31).aspx

WiiMoteLib.dll Hiervoor is er geen installatie nodig, maar je kunt met een rechtermuisklik op reference in het menuutje kiezen voor add reference. Hierna browse je naar de juiste file.

De *Wiimote* gaat communiceren met je applicatie via *Bluetooth*. Het instellen van Bluetooth kun je ook overal op het Internet vinden. Google: Wiimote Bluetooth Ook zijn er verschillende testprogramma's te vinden.

XBOX 360

Bij de ISSD kun je ook een XBOX lenen om daar je XNA-applicatie op te draaien. In de C#/XNA programma is een kleine aanpassing nodig om de applicatie geschikt te maken voor de XBOX. Voer de volgende stappen uit om een game van je laptop naar de Xbox te porten: • Rechtermuisknop klikken op je windowsproject • Selecteer "Create a Copy of Project for Xbox 360"

Zie uitgebreider http://msdn.microsoft.com/en-us/library/bb976061.aspx Even uitproberen. Je moet namelijk in een lan zitten, ook lid zijn van creatorclub enzo.

5.8 XNA en/of gaming bronnen

Let op: XNA wordt niet meer gebruikt voor OIS. Wij gebruiken MonoGame.

Tutorials voor veelgebruikte gaming aspecten (let op: deze zijn gemaakt voor XNA 3.0 dus werken soms net iets anders)

http://www.xnadevelopment.com/tutorials.shtml

http://creators.xna.com

http://creators.xna.com/en-us/Academia

http://www.xbox.com/en-US/kinect

http://depts.washington.edu/cmmr/Research/XNA_Games/XNACS1WorkBook.html

http://dxframework.org/

Game Studio Express[9]

Video Game programming[10]

http://www.garagegames.com/products/torque/x/

https://www.academicresourcecenter.net/curriculum/FacetMain.aspx?FT=Tag&TagList=3&ResultsTitle=Gaming%20and%20Graphics&ShowResults=1

[9] http://msdn2.microsoft.com/en-us/library/bb200104.aspx
[10] http://www.academicresourcecenter.net/curriculum/pfv.aspx?ID=6878

http://www.youtube.com/watch?v=6AKgH4On65A

http://www.youtube.com/watch?v=c_LrVqI6StY

http://graphics.cs.columbia.edu/projects/goblin/

5.9 XNA problemen oplossen

No suitable graphics card found

Het kan gebeuren dat XNA de volgende foutmelding geeft:

Dit probleem is op te lossen door de instelling ”de volledige HiDef API gebruiken” die standaard aan staat te veranderen in ”Use Reach to access a limited API set ....”. Deze instelling is te vinden bij ”Project Properties” op het tabblad ”XNA Game Studio”.

XNA programma kan niet worden uitgevoerd

Het gebruik hiervan op zogenaamde ‘N’ versies van windows 7 kan tot problemen leiden bij het gebruik van XNA. Voor meer info over ‘N’ versies van windows 7, zie: http://en.wikipedia.org/wiki/Windows_7_editions#Sub-editions N versies van windows 7 missen Windows Media Player en de daarbij horende codecs, door EU restricties. Gevolg hiervan is dat XNA programma’s die (bijvoorbeeld) WAV bestanden bevatten niet kunnen compileren. In plaats van een melding dat je WMP/codecs mist, krijg je bij het compileren een cryptische melding over een missende DLL die je, eenmaal gevonden, niet kunt toevoegen aan je project Oplossing voor dit probleem: Windows Media Player downloaden en installeren, zie: http://windows.microsoft.com/en-US/windows/downloads/windows-media-player (Deze linkt door naar deze link: http://www.microsoft.com/downloads/en/details.aspx?FamilyID=31017ed3-166a-4c75-b90c-a6cef9b414c4)

CHAPTER 6

Classes

To do Dit hoofdstuk wordt binnenkort ingevuld.

CHAPTER 7

Constructors

7.1 Opdrachten met constructors

Invaders, Galgje

CHAPTER 8

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 alsvolgt 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-station[1]

Exceptions at MSDN[2]

[1] http://www.csharp-station.com/Tutorial/CSharp/Lesson15
[2] https://msdn.microsoft.com/en-us/library/ms173162.aspx

CHAPTER 9

Graphics

Met de Windows .NET-grafiekbibliotheek kun je lijnen, ellipsen, plaatjes, enz. tekenen. Geschikt voor games of speciale user interfaces.

Graphics-objecten

Kleuren: `class Color`

```
Color myColor;
myColor = Color.Aquamarine;
yColor = Color.Tomato;
```

Pennen: `class Pen`

Brushes: `class SolidBrush`, `HatchBrush`, `LinearGradientBrush`, ...

Lijnen tekenen

Om een lijn te tekenen:

Maak een instantie van een Pen-klasse en zet een paar properties:

```
Pen myPen = new Pen(Color.Red);
myPen.Width = 5;
```

Gebruik het graphics-object om te tekenen. Maak een `Paint Event Handler` aan. De `parameter` **e** bevat informatie over het zogenaamde `Grapics object`:

```
Graphics g = e.Graphics;
```

Teken een lijn:

```
DrawLine(myPen, 1, 1, 45, 65);
```

Methoden van het graphics object

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

CHAPTER 10

Properties

Properties gebruik je om toegang te verlenen tot afgeschermde (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 tijd = sw.seconds;                  // Mag niet
int tijd = sw.Seconds;                  // Mag wel (hoofdletter)
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 tutorials[1]

CodeProject[2]

MSDN[3]

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 properties[4]

[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
[4] http://msdn.microsoft.com/en-us/library/w86s7x04.aspx

CHAPTER 11

Private

11.1 Private

- *Information hiding*: `variabelen` binnen een `class` zijn 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`.

11.2 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.

11.3 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!

11.4 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.

11.5 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 die 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. Focus voor nu vooral op de public en private eigenschappen: de overigen komen pas later in het curriculum aan bod.

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.

11.6 External References

Over private[1]

Techopedia[2]

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

[2] http://www.techopedia.com/definition/3787/encapsulation-c

CHAPTER 12

Separation of GUI and class code

Zorg ervoor dat je geen user interfacecode (zoals bijv. MessageBoxen) in je klassen hebt staan. Gebruik exception handling waar nodig, maar ook niet onnodig.

Werk met `lijsten` (van `objecten`, dus niet van `strings`) en gebruik `casting` om een door de gebruiker geselecteerd `object` weer te pakken te krijgen. Elke `class` heeft een `ToString()` zodat `objecten` goed getoond worden in de `UI`.

CHAPTER 13

File Handling

13.1 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?

13.2 File handling klassen

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

- File
- Directory
- DirectoryInfo
- Path

13.3 Andere bronnen

MSDN[1]

[1] https://msdn.microsoft.com/en-us/library/2kzb96fk.aspx

CHAPTER 14

Enum

14.1 Definitie 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.

14.2 Voorbeelden

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

Dan is mogelijk:

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

CHAPTER 15

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 altijd 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.

CHAPTER 16

XAML

> A good programmer is someone who always looks both ways before crossing a one-way street

Het leerdoel XAML van OIS12 houdt in dat je een user interface voor WPF of Windows Universal kunt maken.

Je kunt dit doen vanuit de designer (Shift+F7) maar ook rechtstreeks in XAML-code. Dit ziet er in het begin misschien eng uit, maar dat went snel.

Tip: als je googelt zet dan het woord "WPF" erbij omdat WPF de eerste techniek was waar XAML bij werd toegepast waardoor er nog steeds veel info op internet over te vinden is.

Liefhebbers kunnen de XAML-editor gebruiken, maar je kunt ook vanuit het design werken. Als je gaat googlen zet dan de zoekterm *WPF* erbij.

16.1 Trucs:

Ik zie geen designer[1]

Tutorial grid-layout (goede manier om controls te alignen)[2]

Advanced topic: styles[3]

[1] http://stackoverflow.com/questions/34088737/visual-studio-2015-update-1-xaml-designer-not-showing-for-uwp
[2] http://www.dailyfreecode.com/code/grid-layout-xaml-212.aspx
[3] https://msdn.microsoft.com/en-us/library/ms745683%28v=vs.110%29.aspx

16.2 Scheiding GUI en code

Eerder is al aangegeven dat het zaak is je code goed gescheiden te houden van de `user interface` (`Forms`). Om te controleren of je dat echt goed hebt gedaan ga je nu een nieuw project maken waarbij je alleen de user interface van je systeem gaat wijzigen.

Maak een `Windows Universal app` aan en maak opnieuw een `User Interface` van een eerder gemaakt programma. Je bestaande klassen met code ga je hergebruiken.

CHAPTER 17

External Resources

Part V

Challenges

CHAPTER 18

Challenge BattleSim

18.1 Voorkomende leerdoelen

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

Zie Chapter 2 voor meer informatie over deze leerdoelen.

18.2 Vereiste voorkennis

OIS11.

Datum	Week 11/12
Versie	1 - Inge van Engeland
Leerdoelen	Class, Property, Constructor

18.3 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 18-1 battlesim

18.4 Opdracht

Programmeer de BattleSim. Figuur 18-1 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.

18.5 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.

CHAPTER 19

Challenge Cube Graphic

19.1 The Cube

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

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

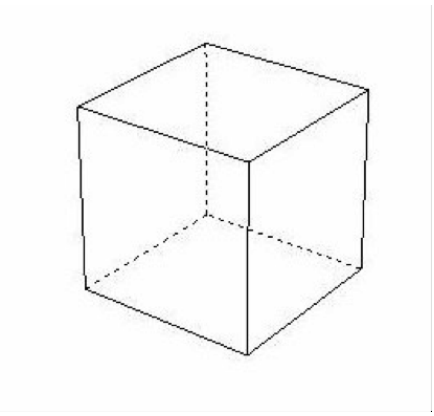

Figure 19-1 cube

CHAPTER 20

Challenge Exception Handling

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 Namen.txt. Het programma leest dit bestand uit.

20.1 Opdracht 1

Voeg C#-code toe die alle excepties van het programma zoals bijvoorbeeld de IndexOutOfRangeException en excepties 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.

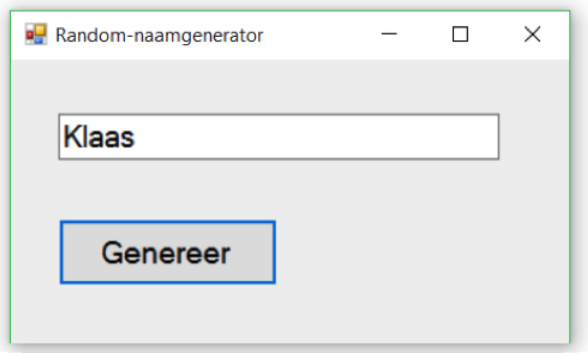

Figure 20-1 naamgenerator

20.2 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.

CHAPTER 21

Challenge Ms PacMan

21.1 Voorkomende leerdoelen

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

Zie Chapter 2 voor meer informatie over deze leerdoelen.

21.2 Vereiste voorkennis

OIS11.

21.3 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 21-2.

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 21-3

.

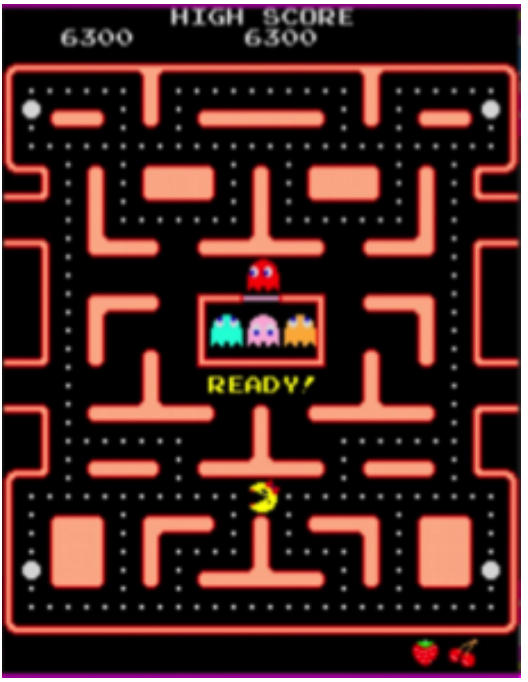

Figure 21-1 pacman

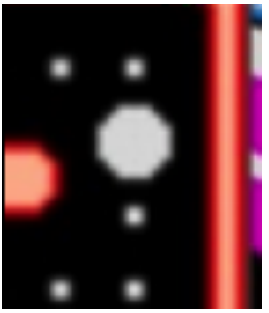

Figure 21-2 pil

Figure 21-3 muur

CHAPTER 22

Challenge File Handling

22.1 Voorkomende leerdoelen

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

22.2 Vereiste voorkennis

OIS11.

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 22-1

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.

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.

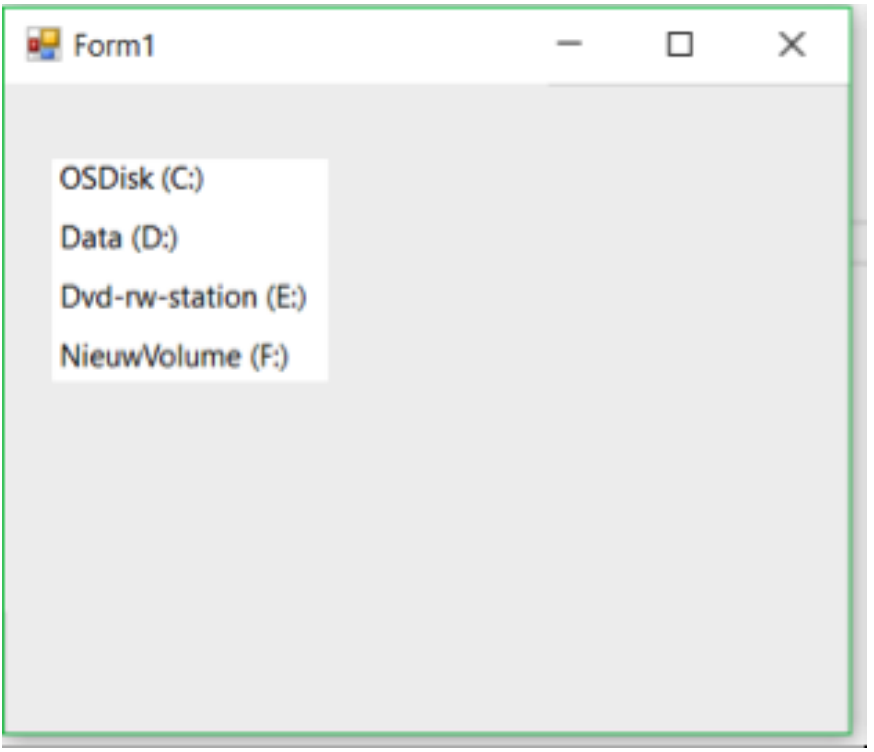

Figure 22-1 Mijn Computer.

CHAPTER 23

Challenge Iedereen kan Schilderen

23.1 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 excepties in je C#-programma kunnen afhandelen.

23.2 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.

23.3 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.

23.4 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.

23.5 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 24

Challenge Super Galgje

24.1 Voorkomende leerdoelen

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

Zie Chapter 2 voor meer informatie over deze leerdoelen.

24.2 Vereiste voorkennis

OIS11.

24.3 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.

CHAPTER 25

Challenge Auto Dagwaarde

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/KM) * 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.

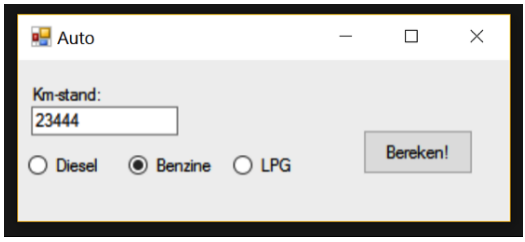

Figure 25-1 dagwaardeberekening

Technische randvoorwaarden

De volgende technische requirements zijn van toepassing:

- 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.

CHAPTER 26

Challenge BankRekening

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

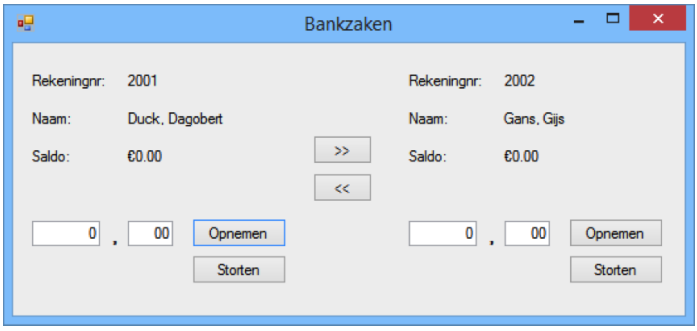

Figure 26-1 bankrekening

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 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.

26.1 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 Contructors 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
    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 ontwerpscherm, 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 saldoinformatie 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 weergeven 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.

26.2 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.

Figure 26-2 niveau

CHAPTER 27

Challenge Woordenzoeker

Op basis van een set woorden (uit een tekstbestand) een woordenzoeker puzzel van een opgegeven breedte x hoogte genereren. Woorden worden verstopt 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 27-1 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, excepties, static, casting

Eventueel: XAML

Bronnen

Sprite Font Generator[1]

Write Text on a Bitmap[2]

Bronnen benodigd bij de extra features

Sound Engine[3]

Database connectie e.d.[4]

[1] 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

Webclient[5]

How to: Request Data Using the WebRequest Class[6]

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

JSON Parser[8]

Variatie / extra features:

Niveau - Feature

* - 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 soundeffects 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 linkerknop 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).

[5] 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-c

[8] https://www.newtonsoft.com/json

* - 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.

CHAPTER 28

Object Oriented Invul-Oefening

Vul de volgende woorden op de juiste plaats in: klasse, instantie, methode, operatie, object, attribuut. Let op: een woord kan meerdere malen in de tekst voorkomen, en soms vul je de meervoudsvorm 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 klassendiagram 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 velden voor AantalLevens en HaarKleur en vervolgens mapt hij de ..... MoveForward en ValDoorNeer naar C#-.....

Als hij de ..... helemaal heeft 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.

CHAPTER 29

Challenge Invaders

To do Dit hoofdstuk is Under Construction (houd de canvasversie er ook bij)

29.1 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.

29.2 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 30

Challenge Galgje

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 conctructor 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.

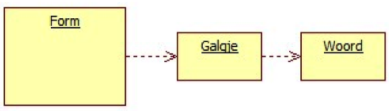

Figure 30-1 galgje

CHAPTER 31

Challenge Dino Game

31.1 Voorkomende leerdoelen

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

Zie Chapter 2 voor meer informatie over deze leerdoelen.

31.2 Vereiste voorkennis

OIS11.

31.3 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 muntjes 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.

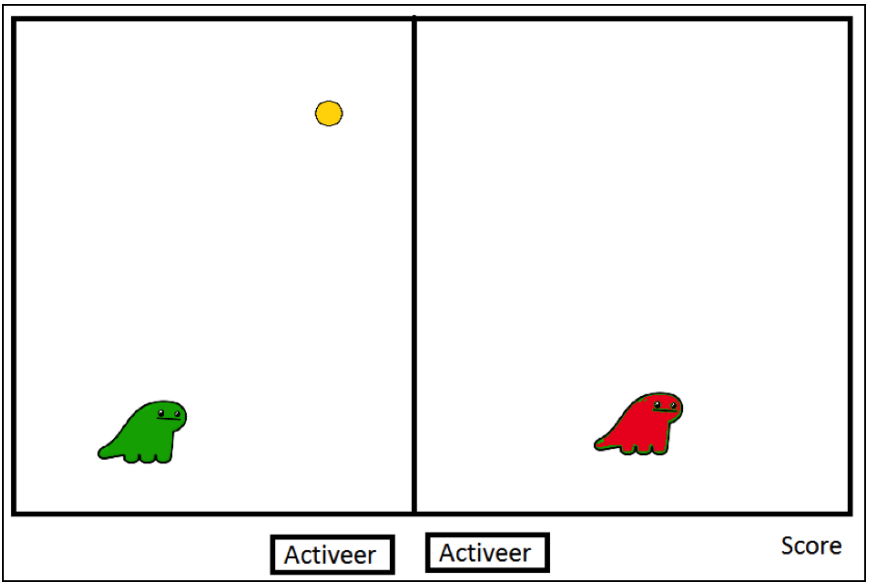

Figure 31-1 dino

CHAPTER 32

Challenge Webshop Spierballetje

32.1 De opdracht

Webshop spierballetje.com verkoopt spullen voor krachttrainers 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.

CHAPTER 33

Challenge Vier op een rij

33.1 Voorkomende leerdoelen

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

Zie Chapter 2 voor meer informatie over deze leerdoelen.

33.2 Vereiste voorkennis

OIS11.

33.3 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 klasse Zet met twee integer property's 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.WriteLines` (loop dan door het arraay 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));
}
```

CHAPTER 34

Challenge Platenmaatschappij

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.

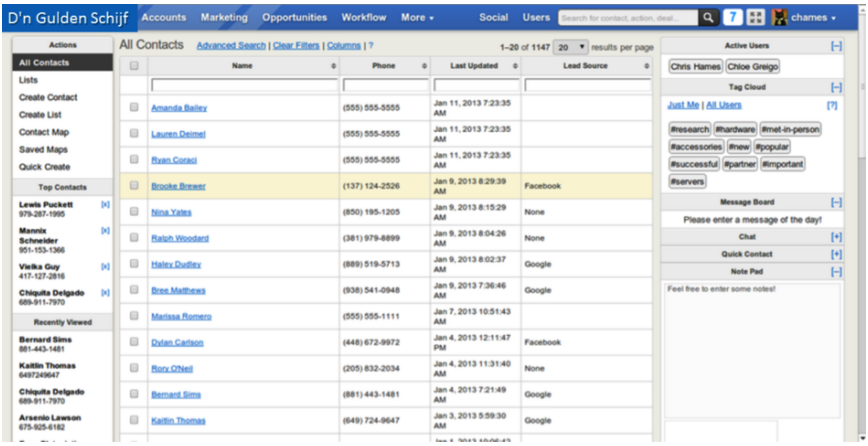

Figure 34-1 platenmaatschappij

CHAPTER 35

External Challenges

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

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

Een ander voorbeeld: codegolf.stackexchange.com[2]

Een site waar allerlei (vaak wiskundige) programmeeruitdagingen op staan is projecteuler.net[3] Door softwaredocenten is Euler-project 54 (genaamd *poker hands*) al genoemd als een voorbeeld van een leuke uitdaging.

[1] https://www.codingame.com/home
[2] http://codegolf.stackexchange.com/
[3] https://projecteuler.net/

CHAPTER 36

Create your own Challenge

Een hele goeie manier om te leren en te kijken of je een onderwerp helemaal snapt is het zelf bedenken van een opdracht. Je helpt ook je medestudenten als je zelf lesmateriaal ontwikkelt voor een nieuwe OIS12-opdracht. Schrijf zelf je eigen opdracht en lever deze in bij de docent. Dan adopteren we jouw idee en leeft jouw eigen opdracht-idee verder als officieel lesmateriaal.

CHAPTER 37

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`.

CHAPTER 38

Challenge Windows Phone

38.1 Voorkomende leerdoelen

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

Zie Chapter 2 voor meer informatie over deze leerdoelen.

38.2 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 drietaal 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 39

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. Kijk vooral eens naar:

Instantiate[1]

Coroutines[2]

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 life 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

[1] http://docs.unity3d.com/ScriptReference/Object.Instantiate.html
[2] http://docs.unity3d.com/Manual/Coroutines.html

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!