1.0.0

H8: Numerieke data

In dit hoofdstuk kijken we naar meer gevorderde operaties die vooral te maken hebben met getallen.

De Math klasse

Berekeningen

Een groot deel van je leven als ontwikkelaar zal bestaan uit het bewerken van variabelen in code. Meestal zullen die bewerkingen de vorm aannemen van berekeningen. De `Math` klasse zal ons hier bij kunnen helpen.

De Math klasse

De Math klasse bevat aardig wat handige methoden. Deze klasse bevat methoden voor een groot aantal typische wiskundige methoden (sinus, cosinus, vierkantswortel, macht, afronden, etc.) en kan je dus helpen om leesbaardere expressies te schrijven.

Stel dat je de derde macht van een variabel `getal` wenst te berekenen. Zonder de Math-klasse zou dat er zo uitzien:

```
double result = getal*getal*getal;
```

Met de klasse kunnen we schrijven:

```
double result = Math.Pow(getal, 3);
```

Het voordeel is dat je dit makkelijk kan aanpassen naar een hogere macht.

De Math klasse ontdekken

Als je in Visual Studio Code `Math` schrijft, gevolgd door een punt `.` krijg je alles te zien wat de Math-klasse kan doen.

Methoden gebruiken

1. Schrijf de Methode zonder argumenten. Bijvoorbeeld `Math.Pow()` (je mag de rode error negeren).
2. Je krijgt nu de help-files te zien van deze methode op MDSDN.
3. Klik desnoods op de pijltjes voor de verschillende versies van deze methode.

PI (π)

Ook het getal Pi (`3.141...`) is beschikbaar in de Math klasse. Dit is geen methode, maar een gewone waarde. Het woordje `const` betekent dat je deze waarde niet kan aanpassen.

Je kan deze als volgt gebruiken in berekeningen:

```
double straal = 5.5;
double omtrek = Math.PI * 2 * straal;
```

Klassiek afronden

"Klassiek" afronden is afronden tot het dichtstbijzijnde getal. Dit doe je met `Math.Round`. Deze methode heeft twee parameters: het getal dat je wil afronden en het aantal cijfers na de komma dat je wil bewaren. Let hierbij goed op: dit berekent een nieuwe waarde (van type `double`) met de gewenste **precisie**, maar zorgt er niet automatisch voor dat deze waarde ook met dat aantal cijfers **getoond** wordt. Anders gezegd: `Math.Round(12.0, 2)` kan exact voorgesteld worden met hooguit twee cijfers na de komma, maar wordt standaard niet getoond met twee cijfers na de komma. Dat laatste behandelen we verder bij stringformattering.

Afronden naar boven of beneden

Naast "klassiek" afronden kan je ook zuiver in één richting afronden. Dan krijg je het dichtstbij gelegen gehele getal waarvoor geldt benadering =< getal (bij naar beneden afronden) of benadering >= getal (bij naar boven afronden). Dit doen we met `Math.Floor` en `Math.Ceiling`. Bijvoorbeeld: `Math.Floor(12.6)` is `12.0`, ook al is 13 een benadering die dichter bij ligt. Dat komt omdat 12 =< 12.6, terwijl 13 > 12.6.

Random

> Kennisclip voor deze inhoud

Random getallen genereren

Random getallen genereren in je code kan nuttig zijn om verschillende scenario's te simuleren, om software te beveiligen of om de gebruiker een interactievere ervaring te geven.

Random generator

De `Random`-klasse laat je toe om eenvoudig willekeurige gehele en komma-getallen te maken. Je moet hiervoor twee zaken doen:

1. Maak *eenmalig* een Random-generator aan: `Random randomgen = new Random();` (wat dit juist wil zeggen zien we in Semester 2).
2. Roep de `Next` methode aan telkens je een nieuw getal nodig hebt, bijvoorbeeld: `int mijnGetal = randomgen.Next();` De aanroep van de methode `Next()` zal een geheel getal willekeurig genereren.

De eerste stap dien je dus maar één keer te doen. Vanaf dan kan je telkens aan de generator een nieuw getal vragen m.b.v. `Next`.

Volgende code toont bijvoorbeeld 3 random getallen op het scherm:

```
Random mygen = new Random();
int getal1 = mygen.Next();
int getal2 = mygen.Next();
int getal3 = mygen.Next();
Console.WriteLine(getal1);
Console.WriteLine(getal2);
Console.WriteLine(getal3);
```

Next mogelijkheden

Je kan de `Next` methode ook 2 waarden meegeven, namelijk de grenzen waarbinnen het getal moet gegenereerd worden. De tweede waarde wordt nooit gegenereerd. Wil je dus een getal tot en met 10 dan schrijf je 11, niet 10.

Enkele voorbeelden:

```
Random somegenerator = new Random();

int a = somegenerator.Next(0,10);  //getal tussen 0 tot en met 9
int b = somegenerator.Next(5,101);  //getal tussen 5 tot en met 100
int c = somegenerator.Next(0,b);  //getal tussen 0 tot en met het getal dat de lijn ervoor w
```

Genereer kommagetallen met NextDouble

Met de `NextDouble` methode kan je kommagetallen genereren tussen `0.0` en `1.0` (1.0 zal niet gegenereerd worden).

Wil je een groter kommagetal dan zal je dit gegenereerde getal moeten vermenigvuldigen naar de range die je nodig hebt. Stel dat je een getal tussen 0.0 en 10.0 nodig hebt, dan schrijf je:

```
Random myran = new Random();
double randomgetal= myran.NextDouble() * 10.0;
```

Casting en conversie

Casting en conversie

> Kennisclip basis (meer kennisclips lager op de pagina)

Wanneer je de waarde van een variabele wil toekennen aan een variabele van een ander type mag dit niet zomaar. Volgende code zal bijvoorbeeld een error geven, omdat aan de linkerkant staat dat je een geheel getal wil bijhouden (door middel van `int`) en aan de rechterkant een kommagetal staat:

```
int age = 4.3;
```

Een kommagetal past niet in het geheugen voorzien voor een geheel getal: als er toch een zinvolle omzetting mogelijk is, zal je moeten **converteren of casten**.

Dit kan op o.a. volgende manieren:

- Via casting: de (klassieke) manier die ook werkt in veel andere programmeertalen.
- Via de `Convert` klasse. Deze staat omzettingen toe die handig zijn, maar niet op het niveau van de taal zijn vastgelegd.

Casting

Een klassieke, in veel talen voorziene manier om date om te zetten is **casten**. Hierbij dien je aan de compiler te zeggen: *"Volgende variabele die van het type x is, moet aan deze variabele van het type y toegekend worden.* ***Ik besef dat hierbij data verloren kan gaan, maar zet de variabele toch maar om naar het nieuwe type, ik draag alle verantwoordelijkheid voor het verlies.***" Je kan bijvoorbeeld bovenstaand kommagetal enkel omzetten naar een geheel getal als je de cijfers na de komma verloren laat gaan. In het Engels betekent *to cast* "in een vorm gieten".

Wat is casting

Casting heb je nodig om een waarde van een bepaald type om te zetten naar een waarde van een ander type. Stel dat je een complexe berekening hebt waar je werkt met verschillende types (bijvoorbeeld int, double en float). Door te casten kan je het soort bewerkingen sturen. Je gaat namelijk bepaalde types even als andere types gebruiken.

Het is belangrijk in te zien dat het casten van een variabele naar een ander type enkel een gevolg heeft tijdens het uitwerken van de expressie waarbinnen je werkt. Het casten op zich verandert niets aan de variabele of waarde zelf.

Casting duid je aan door voor de variabele of literal het datatype tussen haakjes te plaatsen naar wat het omgezet moet worden:

```
int mijngetal = (int) 3.5;
Console.WriteLine(mijngetal);
```

of

```
double kommagetal = 13.8;
int kommaNietWelkom = (int) kommagetal;
Console.WriteLine(kommaNietWelkom);
```

Narrowing

> Kennisclip narrowing en widening

Casting gebruik je vaak als je een variabele wil gebruiken als een ander type dat deze waarde niet kan bevatten. We moeten dan aan **narrowing** doen, letterlijk het versmallen van de data.

Bekijk eens het volgende voorbeeld:

```
double var1;
int var2;

var1 = 20.4;
var2 = var1;
```

Dit zal niet gaan. Je probeert namelijk een waarde van het type double in een variabele van het type int te steken. Dat gaat enkel als je informatie weggooit. Je moet aan *narrowing* doen.

Dit gaat enkel als je expliciet aan de compiler zegt: het is goed, je mag informatie weggooien, ik begrijp dat en zal er rekening mee houden. Dit noemen we **expliciete casting**.

En je doet dit door voor de variabele die dienst moet doen als een ander type, het nieuwe type, tussen ronde haakjes te typen, als volgt:

```
double var1;
int var2;

var1 = 20.4;
var2 = (int) var1;
```

Het resultaat in `var2` zal `20` zijn (alles na de komma wordt bij casting van een double naar een int weggegooid).

> Merk op dat `var1` nooit van datatype is veranderd; enkel de inhoud ervan (`20.4`) werd eruit gehaald, omgezet ("gecast") naar `20` en dan aan `var2` toegewezen dat enkel `int` aanvaardt.

Widening

Casting kan je ook gebruiken als je aan **widening** doet (een kleiner type in een groter type steken), als volgt:

```
int var1;
double var2;

var1 = 20;
var2 = var1;
```

Deze code zal zonder problemen gaan. `var2` zal de waarde `20.0` bevatten. De inhoud van `var1` wordt *verbreed* naar een `double`, eenvoudigweg door er een kommagetal van te maken. Er gaat **geen** inhoud verloren. Dit wordt ook **impliciete casting** genoemd. Er bestaan scenario's waarin je dit toch expliciet wil doen, maar die komen minder vaak voor en zullen we benoemen als we ze tegenkomen.

Conversie

> Kennisclip Convert klasse

Casting is een in de taal ingebakken manier van data omzetten, die vooral zeer nuttig is daar deze ook werkt in andere C#-related programmeertalen zoals C, C++ en Java. Om te weten hoe deze omzettingen gebeuren, kan je kijken in de handleiding van de taal C# zelf.

Echter, .NET heeft ook methoden die je kunnen helpen om data van het ene type naar het andere te brengen. Deze methoden zitten binnen de **`Convert`**-klasse.

Het gebruik hiervan is zeer eenvoudig. Enkele voorbeelden:

```
int getal= Convert.ToInt32(3.2); //double to int
double anderGetal= Convert.ToDouble(5); //int to double
bool isWaar= Convert.ToBoolean(1); //int to bool
int userAge= Convert.ToInt32("19"); //string to int
int ageOther= Convert.ToInt32(anderGetal); //double to int
```

Je plaatst tussen de ronde haakjes de variabele of literal die je wenst te converteren naar een ander type. Merk op dat naar een `int` converteren met `.ToInt32()` moet gebeuren. Om naar een `short` te converteren is dit met behulp van `.ToInt16()`. Het voordeel van deze werkwijze is dat ze vastgelegd kan worden door de programmeur en dat ze vaak meer "intuïtieve" omzettingen doet dan een cast. En, als de omzettingen je niet bevallen, kan je een eigen klasse `MyConvert` schrijven die de omzettingen doet zoals je zelf wil. **Het zijn namelijk gewoon methodes.**

Je kan alle conversie-mogelijkheden hier bekijken.

Oefeningen

Inleiding

Deze oefeningen maak je als statische methoden van een klasse `NumeriekeData`. Je kan elk van deze methoden uitvoeren via een keuzemenu, zoals bij vorige hoofdstukken.

H8-LengteOppervlakteVolume

Leerdoelen

- Werken met machten

Functionele analyse

Schrijf een programma om, gegeven de lengte van een zijde, de oppervlakte van een vierkant met die zijde en het volume van een kubus met die zijde te berekenen. Je mag maximum vier keer gebruik maken van de variabele met daarin de lengte van de zijde, inclusief de initialisatie.

Technische analyse

De oppervlakte bepaal je door de zijde met zichzelf te vermenigvuldigen. Bijvoorbeeld, een zijde van 10m betekent een oppervlakte van 10m * 10m = 100m². Het volume bepaal je door nog een keer met de zijde te vermenigvuldigen. Een zijde van 10m betekent dus een oppervlakte van 1000m³. Maak gebruik van een methode uit `Math`.

Noem je methode `LengteOppervlakteVolume`.

Voorbeeldinteractie

```
Hoe lang is de zijde in meter?
> 10
De lengte is: 10m
De oppervlakte is: 100m²
Het volume is: 1000m³
```

Oefening: H8-schaar-steen-papier

Leerdoelen

- een taak herhaaldelijk uitvoeren met een lus

Functionele analyse

Maak een applicatie waarbij de gebruiker steen-schaar-papier met de computer kan spelen. De gebruiker kiest telkens steen, schaar of papier en drukt op enter. Vervolgens kiest de computer willekeurig steen, schaar of papier.

Vervolgens krijgt de winnaar 1 punt:

- Steen wint van schaar, verliest van papier
- Papier wint van steen, verliest van schaar
- Schaar wint van papier, verliest van steen
- Indien beide hetzelfde hebben wint niemand een punt.

De eerste (pc of gebruiker) die 10 punten haalt wint.

Noem je methode `SchaarSteenPapier`.

Technische analyse

- Genereer een willekeurig getal tussen 1 en 3 om de computer te laten kiezen.
- Teken een flowchart!

Voorbeeldinteractie

```
Maak een keuze:
1 voor schaar
2 voor steen
3 voor papier
> 2
De computer kiest steen!
Niemand wint deze ronde!
Jij hebt 0 punten, de computer heeft 0 punten.
Maak een keuze:
1 voor schaar
2 voor steen
3 voor papier
> 1
De computer kiest papier!
Jij wint deze ronde!
Jij hebt 1 punt, de computer heeft 0 punten.
Maak een keuze:
1 voor schaar
2 voor steen
3 voor papier
> 1
De computer kiest steen!
De computer wint deze ronde!
Jij hebt 1 punt, de computer heeft 1 punt.
```

(Helemaal op het einde)

```
Jij hebt 10 punten, de computer heeft 8 punten.
Jij bent gewonnen!
```

of

```
Jij hebt 8 punten, de computer heeft 10 punten.
De computer is gewonnen!
```

Oefening: TextCell met willekeurige getallen

Maak voor deze oefening een aparte klasse TextCellMetRandom. Kopieer hierin de code van je klasse TextCell en pas daarna pas aan.

Functionele analyse

We willen onze spreadsheets gebruiken voor simulaties. Hiervoor hebben we willekeurige data nodig.

Technische analyse

Zorg ervoor dat een cel in TextCell ook een formule van de vorm `=rand()` kan bevatten. Dit initialiseert de cel op een willekeurige waarde tussen 1 en 10.

Oefening: TextCell met kommagetallen

Maak voor deze oefening een aparte klasse TextCellMetKomma. Kopieer hierin de code van je klasse TextCell en pas daarna pas aan.

Functionele analyse

We willen complexere berekeningen doen in TextCell. Zorg ervoor dat kommagetallen ondersteund worden, maar wel steeds getoond worden met *maximum* twee cijfers na de komma.

Technische analyse

Pas op de juiste plaatsen alle code die een `int` veronderstelt aan naar een `double`. Zorg ervoor dat, na het uitrekenen van alle formules, alle cellen klassiek afgerond worden tot 2 cijfers na de komma.