Programma sturing

Les 3: Programma sturing

De if…else instructie

De selectie instructie in C# ziet er als volgt uit:

```
if (Boolean expressie)
    Instructie of blok van instructies
else
    Instructie of blok van instructies
```

Het **else** gedeelte is optioneel. Een veel voorkomende vorm van de **if** instructie is:

```
If (==5)
    Console.WriteLine("Thank God, I Finally became 5.");
```

In het bovenstaande voorbeeld, zal de console boodschap alleen afgedrukt worden als de expressie i == 5 waar (true) is. Als je echter ook actie wilt ondernemen als de expressie niet waar is dan ziet de instructie er bijvoorbeeld als volgt uit:

```
If (==5)
    Console.WriteLine("Thank God, I Finally became 5.");
else
    Console.WriteLine("Missed…When will I become 5?");
```

De eerste boodschap wordt alleen afgedrukt als i gelijk is aan 5. In alle andere gevallen zal de tweede boodschap getoond worden in het console venster.

Je kunt een blok van instructies plaatsen tussen open en sluit accolades {}:

```
If (i==5)
{
    j = i * 2;
    Console.WriteLine("Thank God, I Finally became 5.");
}
else
{
    j = i / 2;
    Console.WriteLine("Missed…When will I become 5?");
}
```

Programma sturing

De switch…case instructie

Wanneer er meerdere specifieke controles moeten worden uitgevoerd is het verstandiger om de switch…case instructie in C# te gebruiken. De algemene structuur van de switch…case instructie is als volgt:

```
switch(integral or string expression)
{
   case constant-expression;
         statements
         breaking or jump statement
   // andere case blokken
   …
   default:
          statements
          breaking or jump statement
}
```

Het kost minder tijd om een switch instructie te gebruiken dan meerdere if instructies.

Bekijk het onderstaand voorbeeld:

```
switch(input)
{
   case 1:                          // als input 1 is
         Console.WriteLine("Je hebt 1 ingevoerd.");
         Break;
   case 2:                          // als input 2 is
         Console.WriteLine("Je hebt 2 ingevoerd.");
         Break;
   case 3:                          // als input 3 is
         Console.WriteLine("Je hebt 3 ingevoerd.");
         Break;
  default:
         Console.WriteLine("Je hebt geen 1, 2 of 3 ingevoerd.");
         Break;
}
```

Programma sturing

De for lus

Lussen worden gebruikt om de zelfde instructies meerdere keren te herhalen tot een bepaalde conditie (voorwaarde) bereikt is.

De meest gebruikte lusvorm in C# is de for-lus. De basis structuur voor deze is het zelfde als in de talen Java en C/C++ en ziet er als volgt uit:

```
for(instructie; conditie; verhogen/verlagen)
    instructies of blok van instructies
   omsloten tussen {} accolades.
```

Laten we gebruik maken van de for lus om de gehele getallen 1 tot en met 10 in het console venster af te laten drukken. De instructie ziet er dan als volgt uit:

```
for(int i=1; i<=10; i++)
{
    Console.WriteLine("In de lus, de waarde van i is {0}.", i);
}
```

Bij het starten wordt de integer variabele i geïnitialiseerd met de waarde 1. De instructies in de lus worden uitgevoerd zolang de voorwaarde (i<=10) waar blijft. De waarde van i wordt verhoogd (i++) met 1 elke keer als de lus start.

Je kunt de instructies break en continue gebruiken om het uitvoer verloop van een lus te beïnvloeden.

```
for(int i=1; i<=10; i++)
{
    if(i>5)
    {
        break;
    }
    Console.WriteLine("In de lus, de waarde van i is {0}.", i);
}
```

De lus hierboven beëindigt zodra de waarde van i groter wordt dan 5. Als er instructies zijn na de break opdracht, dan moet break ingesloten worden onder een voorwaarde, anders zullen de andere instructies nooit bereikt worden en zal de compiler een waarschuwing genereren.

```
for(int i=1; i<=10; i++)
{
    Break;   // waarschuwing, Console.WriteLine(i); is onbereikbare code

    Console.WriteLine(i);
}
```

De opdracht negeert continue de rest van de huidige iteratie en gaat verder bij de volgende iteratie.

```
for(int i=1; i<=10; i++)
{
     if(i==5)
    {
          break;
    }
    Console.WriteLine("In de lus, de waarde van i is {0}.", i);
}
```

Wat zou de uitvoer zijn van het stuk code hierboven.

De do…while lus

De algemene structuur voor de do…while lus is als volgt:

```
do
     Intructie of blok van instructies
While(boolean expressie);
```

Het programma om de integers 1 tot en met 10 op het scherm af te drukken is als volgt:

```
int i=1;
do
{
     Console.WriteLine("In de lus, waarde van i is {0}.", i);
     I++;
} While(i<=10);
```

Deze lus vorm wordt echter wel afgeraden om te gebruiken daar de instructies eerste uitgevoerd worden zonder te controleren.

Programma sturing

De while lus

De while lus is bijna het zelfde als de do…while lus. Echter bij deze wordt er eerst gecontroleerd voor dat de instructies in de lus worden uitgevoerd.

Het programma om de integers 1 tot 10 op het console venster af te drukken is als volgt:

```
int i=1;
while(i<=10)
{
    Console.WriteLine("In de lus, waarde van i is {0}.", i);
    i++;
}
```

Array Declaraties

Een *arra*y is een verzameling van waarden van het zelfde datatype. In C# zijn arrays van het type referentie. Elke array in C# is een object en geërfd van de klasse **System.Array**. Een array wordt als volgt gedeclareerd:

```
<data type> [] <identifier> = new <data type>[<size of array>];
```

Als we een array definiëren die 10 integers kan bewaren, dan ziet de instructie er zo uit:

```
int [] integers = new int[10];
```

De grote van een array staat vast en moet gedefinieerd zijn voor dat de array gebruikt kan worden. Je kunt variabelen gebruiken om de grote van een array te definiëren.

```
int [] size = 10;
int [] integers; = new int[size];
```

Je kunt de declaratie en initialisatie ook in gescheiden stappen doen:

```
int [] integers;
integers = new int[10];
```

Het is ook mogelijk een array te definiëren door de waarden in een lijst toe te voegen aan de array:

```
int [] integers = {1, 2, 3, 4, 5};
```

Programma sturing

Toegang krijgen tot waarden in een array

Om toegang te krijgen tot de waardes in een array, gebruikt men een index operator [int index].

De index waarden in C# beginnen vanaf 0.

Dus bij een array met 5 elementen, heeft het eerste element een index van 0 en het laatste element een index van 4. Het onderstaand voorbeeld geeft aan hoe het 3de element in een array bereikt wordt:

```
int [] intArray = {10, 20, 30, 50, 50};
int x =intArray[2];
```

Het onderstaand programma fragment illustreert het gebruik van een array:

```
// Demonstratie van arrays in C#

static void Main()
{
    // declareer en initialiseer een array van het type integer
    Int [] integers = {3, 7, 2, 14, 65};
    // ga door de array en druk elk element af in het console venster
    for(int i=0; i<5; i++)
    {
        Console.WriteLine(integers[i]);
    }
}
```

De bovenstaande programma code is vrij eenvoudig, maar de grootte van de arrays is vastgelegd *(hard-coded)* in de lus.

Doordat array's erfen van de klasse System.Array, waar een heleboel bruikbare eigenschappen en methods in zitten die we kunnen gebruiken. Door van deze gebruik te maken kunnen we het programma ook als volgt schrijven:

```
for(int i=0; i<integers.Length; i++)
{
    Console.WriteLine(integers[i]);
}
```

Deze versie is flexibeler en kan worden toegepast op elke array van elke grote van elk datatype.

De foreach lus

Er is een andere methode om arrays of collecties te doorlopen. De foreach lus. De basis structuur van deze instructie is alsvolgt:

```
// Demonstratie van arrays in C#

static void Main()
{
    // declareer en initialiseer een array van het type integer
    int [] integers = {3, 7, 2, 14, 65};
    // ga door de array en druk elk element af in het console venster
    foreach(int i in integers)
    {
        Console.WriteLine(i);
    }
}
```

Opmerkingen:

- De variabele (i) die de individuele waarden van de array elementen bewaart, is **read only**. Dit wil zeggen dat je met foreach alleen de array kunt doorlopen maar niet de waarden wijzigingen van de elementen.
- De foreach instructie kun je gebruiken om arrays of collecties te doorlopen. Collecties zijn classes, structs of interfaces.
- De string class is ook een collectie van karakters.

```
static void Main();
{
    string name = "Juancho Noob";
    foreach(char ch in name)
    {
        Console.WriteLine(ch);
    }
};
```

Dit voorbeeld drukt elke letter van de variabele name op een aparte regel af.

List

De System.Collection.ArrayList klasse is vergelijkbaar met arrays, echter deze kan elementen van elk datatype bewaren. Bij een ArrayList hoeft niet te worden aangegeven wat de grote van de verzameling is (zoals bij de simpele array's). De grote van de ArrayList groeit dynamisch wanneer het aantal elementen die het bevat veranderen. Als het aantal elementen meer of minder wordt, verandert de ArrayList de capaciteit van de array naar gelang nodig is door een nieuwe array te maken en de oude waarden te copiëren in de nieuwe. De *Grootte* van de ArrayList is het aantal elementen die werkelijk aanwezig zijn terwijl de *Capaciteit* het aantal elementen aan geeft die de ArrayList kan bevatten zonder een nieuwe array te moeten maken (instantiëren). Een ArrayList kan op de volgende manier worden gemaakt:

```
ArrayList list = new ArrayList();
```

Je kunt de Capaciteit van de ArrayList initialiseren door een integer waarde mee te geven aan de constructor. *(In een latere les wordt precies uitgelegd wat een constructor is)*

```
ArrayList list = new ArrayList(20);
```

Je kunt ook een ArraList maken met een andere soort verzameling door deze verzameling mee te geven als parameter in de constructor.

```
ArrayList list = new ArrayList(eenVerzameling);
```

We kunnen elementen toevoegen aan een list door gebruik te maken van de Add() method *(method is een functie, wordt later besproken)*. De Add() method neemt een object van het type object als parameter.

```
list.Add(45);
list.Add(87);
list.Add(12);
```

Dit zal de drie getallen toevoegen aan ArrayList. Nu kun je door de elementen lopen met een foreach lus.

```
static void Main()
{
    ArrayList list = new ArrayList();
    list.Add(45);
    list.Add(87);
    list.Add(12);
    foreach(int num in list)
    {
        Console.WriteLine(num);
    }
}
```

Deze code geeft de volgende uitvoer in het console venster:

```
45
87
12
Press any key to continue
```

De ArrayList klasse heft ook een index eigenschap waarmee toegang tot elementen wordt gegeven doormiddel van de index operatoren [], net als met een simpele array.

```
static void Main()
{
    ArrayList list = new ArrayList();
    list.Add(45);
    list.Add(87);
    list.Add(12);
    for(int i=0; i<list.Count; i++)
    {
        Console.WriteLine(list[i]);
    }
}
```

Programma sturing

Hieronder volgt een lijst van enkele belangrijke eigenschappen en methoden van de ArrayList klasse:

Property or method	Omschrijving
Capacity	Gets of Sets het aantal elementen dat de ArrAyList kan bevatten
Count	Vraagt het exacte aantal elementen op in de ArrayList
Add(object)	Voegt een element toe aan het einde van de ArrayList
Remove(object)	Verwijdert een element van de ArrayList
RemoveAt(int)	Verwijderd een element op een specifieke locatie van de ArrayList
Insert(int,object)	Voegt een object toe aan de ArrayList op een specifieke locatie
Clear()	Verwijdert alle elementen van de ArrayList
Contains(object)	Geeft een booleaanse waarde terug die aangeeft of het gezocht element in de ArrayList zit.
CopyTo()	Kopieert de elementen van de ArrayList naar een array die als parameter is meegegeven
IndexOf(object)	Geeft de **zero based** index van het eerste voorkomen van het object in de ArrayList
LastIndexOf(object)	Geeft de zero based index van de laatste keer dat een object in de ArrayList voorkomt
ToArray()	Geeft een array van het type object terug dat alle elementen bevat van de ArrayList
TrimToSize()	Past de capaciteit van de ArrayList aan, aan de werkelijke hoeveelheid elementen in de ArrayList

Enum

Een enum is handig te combineren met cases, zeker als je een gelimiteerd aantal mogelijkheden wilt toestaan. Een enum (kort voor enumeration) is een set aan namen waar je een groepsnaam aan hangt, en elke naam binnen deze groep heeft een unieke waarde. Bijvoorbeeld:

```
enum PlayerState
{
        ALIVE,          //heeft in principe waarde 0
        DEAD,           //1
        DANCING,        //2
        JUMPING         //3
}
```

Programma sturing

Dit kan je handig combineren met een switch, als je een variabele van het type PlayerState als argument gebruikt. Je kunt dus ook variabelen maken met het type van een enum, en die variabelen accepteren dan alleen waardes die overeenkomen met een van de dingen uit de enum.

Bijvoorbeeld:

PlayerState state = PlayerState.ALIVE;