Object georiënteerd programmeren

Tom Peeters
Jesse Viskens
Robin Verbruggen

C# OOP

Generics

Generics

- Type safety
- Minder code en makkelijker herbruikbaar
- Betere performance

Generic Method

```
public static void ToonArray(int[] array)
{
    foreach (var i in array)
    {
        Console.WriteLine(i);
    }
}

public static void ToonArray(string[] array)
{
    foreach (var i in array)
    {
        Console.WriteLine(i);
    }
}
```

Generic Method

```
public static void ToonArray<T>(T[] array)
{
    foreach (T i in array)
    {
        Console.WriteLine(i);
    }
}
```

Generic Type

```
struct Location<T>
{
    public T X;
    public T Y;
    public T Z;
}
```

```
static void Main(string[] args)
{
    Location<int> plaats;
    plaats.X = 34;
    plaats.Y = 22;
    plaats.Z = 56;

    Location<double> plaats2;
    plaats2.X = 34.5;
    plaats2.Y = 22.2;
    plaats2.Z = 56.7;

    Location<string> plaats3;
    plaats3.X = "naast de kerk";
    plaats3.X = "links van de bakker";
    plaats3.Z = "onder het hotel";

}
```

Generic Class

```
public class Meting<T>
{
    private T waarde;
    //constructor
    public Meting(T waardein)
    {
        waarde = waardein;
    }
    //Public property
    public T Waarde { get { return waarde; } set { waarde = value ;} }
    public override string ToString()
    {
        return "Deze meting heeft als waarde:"+ waarde.ToString();
    }
}
```

```
Meting<int> m1 = new Meting<int>(44);
Console.WriteLine(m1);

Meting<string> m2 = new Meting<string>("slechte meting");
Console.WriteLine(m2);
```

Meerdere Types

```
class DataBewaarder<Type1, Type2>
{
    private Type1 waarde1;
    private Type2 waarde2;
    //Constructor
    public DataBewaarder(Type1 w1, Type2 w2)
    {
        waarde1 = w1;
        waarde2 = w2;
    }
    public override string ToString()
    {
        return "Waarde1: " + waarde1 + "\nWaarde2: " + waarde2;
    }
}
```

```
DataBewaarder<int, string> d1 = new DataBewaarder<int, string>(4, "Ok");
```

Constraints

```
public class Wijziging<T> where T : IComparable
{
    public T vorigewaarde;
    public T huidigewaarde;
    public Wijziging(T vorig, T huidig)
    {
        vorigewaarde = vorig;
        huidigewaarde = huidig;
    }

    public bool IsGestegen()
    {
        if (huidigewaarde.CompareTo(vorigewaarde) > 0)
            return true;
        else return false;
    }
}
```

```
Wijziging<double> w = new Wijziging<double>(3.4, 3.65);
Console.WriteLine(w.IsGestegen());
```

Mogelijke constraints

- Enkel value types
- Enkel klassen
- Moet default constructor hebben
- Moet overerven van een bepaalde klasse

Collections – Nadelen van Arrays

- Grootte Array
- Array te klein, kopieren naar grotere Array
- By Reference & By Value

Collections – Geavanceerde Arrays

- Collecties zijn speciale objecten waarin een reeks van andere objecten kan worden gerefereerd
- Er moet eerst een instantie van een collectie aangemaakt worden
- Collecties hebben properties die onder andere het aantal objecten in collectie weergeeft
- Collecties hebben methoden om onder andere objecten toe te voegen en verwijderen
- Daar collecties beschreven zijn in klassen kan je de typische object-georiënteerde eigenschappen hierop toepassen: je kan collecties overerven, encapsuleren (composiet), etc

Collections - Voorbeelden

- ArrayList
- BitArray
- HashTable
- Queue
- SortedList
- Stack

Namespaces

- System.Collections
- System.Collections.Generic

Type Safety

Een niet-generieke collection is **niet type-safe**. Een generieke collection is dat wel.

In dit niet-generieke voorbeeld zal de laatste lijn tijdens uitvoer altijd een Exception geven.

```
ArrayList nietGeneriekeList=new ArrayList();
string naam= "Tim";
nietGeneriekeList.Add(naam);
int leeftijd = (int)nietGeneriekeList[0];
```

Type Safety

Bij een generieke collection zal er bij soortgelijk code een compiler-error optreden (gedrag dat meestal wenselijk is) en de code zal dus niet gecompileerd kunnen worden.

```
List<string> generiekeList=new List<string>();
string naam= "Tim";
generiekeList.Add(naam);
int leeftijd = (int)nietGeneriekeList[0]; // compilererror: cannot convert type string to int
```

Generic Collections

- Strongly typed
- Gebruik is quasi identiek aan niet-generieke collections
- List<T>

Queue<T> Collection

- FIFO (First In First Out)
- `Enqueue(T item)`: Voeg een item achteraan de queue toe
- `Dequeue()`: geeft het eerste element in de queue terug en verwijdert dit element vervolgens uit de queue
- Peek(): geeft het eerste element weer zonder het te verwijderen

Queue<T> Collection

```
Queue<string> theQueue =  new Queue<string>();
theQueue.Enqueue("I arrived here first!");
theQueue.Enqueue("I arrived second.");
theQueue.Enqueue("I'm last");

Console.WriteLine(theQueue.Dequeue());
Console.WriteLine(theQueue.Dequeue());
```

```
I arrived here first!
I arrived here second.
```

Stack<T> Collection

- LIFO (Last In First Out)
- `Push(T item)`: plaats een nieuw item bovenop de stapel
- `Pop()`: geeft het bovenste element in de stack terug en verwijdert vervolgens dit element van de stack
- Peek(): geeft het bovenste element in de stack terug zonder het te verwijderen

Stack<T> Collection

```
Stack<string> theStack =  new Stack<string>();
theStack.Push("I arrived here first!");
theStack.Push("I arrived second.");
theStack.Push("I'm last");

Console.WriteLine(theStack.Pop());
Console.WriteLine(theStack.Pop());
```

```
I'm last
I arrived second.
```

Dictionary Collection

- Key (uniek)
- Value

Dictionary Collection

```
Dictionary<int, string> customers = new Dictionary<int, string>();
customers.Add(123, "Tim Dams");
customers.Add(6463, "James Bond");
customers.Add(666, "The beast");
customers.Add(700, "James Bond");
```

```
foreach (var item in customers)
{
    Console.WriteLine(item.Key+ "\t:"+item.Value);
}
```

```
Console.WriteLine(customers[123]);
```

```
Dictionary<string, Student> studentenLijst = new Dictionary<string, Student>();
Student stud= new Student();
stud.Naam= "Tim";stud.Leeftijd=24;
studentenLijst.Add("AB12",stud);
Console.WriteLine(studentenLijst["AB12"].Leeftijd);
studentenLijst.Remove("AB12");
```