C# Les 10 – Lijsten manipuleren & oefeningen

Als je de naam van een lijst typt in Visual Studio zie je een lijst van opties (waaronder Add, Count en RemoveAt: die kennen we al). We bekijken enkele handige methodes die je kan gebruiken voor lijsten. De methodes die we bespreken werken allemaal volgens hetzelfde principe, met een “naamloze methode”.

Find()

Met Find kunnen we een voorwaarde meegeven. De lijst wordt van voor naar achter doorlopen en de voorwaarde zal één voor één voor elk item gecontroleerd worden. Het eerste item in de lijst dat aan de voorwaarde voldoet wordt teruggegeven.

Find geeft dus het eerste element in een lijst terug dat aan een voorwaarde voldoet.

```
Dranken = new List<Drank>
{
    new Drank("Koffie", 2.5F),
    new Drank("Thee", 3),
    new Drank("Kleine soep", 4),
    new Drank("Grote soep", 6)
};

Drank eersteDureDrank = Dranken.Find(drank => drank.Prijs > 3);

Console.WriteLine(eersteDureDrank.Naam);

// output: Kleine soep
```

De schrijfwijze hierboven is misschien wat eigenaardig, met het pijltje (=>). Een pijltje wordt gebruikt om een mini-methode zonder naam te maken (een “naamloze methode”). drank is het argument van de naamloze methode. Het belangrijkste is dat je begrijpt dat de methode-body (drank.Prijs > 3) voor elk item in de lijst gecontroleerd wordt totdat het eerste element waarvoor dit geldt gevonden wordt.

Als de Find methode niets vindt wordt de waarde “null” teruggegeven.

FindAll()

In plaats van het eerste element op te vragen kan je ook al de elementen opvragen die aan een bepaalde voorwaarde voldoen. FindAll werkt gelijkaardig als Find (ook met een naamloze methode) en geeft een lijst terug.

```
Dranken = new List<Drank>
{
```

```
    new Drank("Koffie", 2.5F),
    new Drank("Thee", 3),
    new Drank("Kleine soep", 4),
    new Drank("Grote soep", 6)
};

var DureDranken = Dranken.FindAll(drank => drank.Prijs > 3);

foreach(Drank d in DureDranken)
{
    Console.WriteLine(d.Naam);
}

/*
output:
Kleine soep
Grote soep
*/
```

RemoveAll()

RemoveAll werkt ook met zo'n naamloze methode, op dezelfde manier als Find en FindAll. Anders dan Find en FindAll gaat RemoveAll de lijst waarop de methode wordt aangeroepen veranderen: al de elementen die aan de voorwaarde voldoen worden uit de lijst gehaald.

```
Dranken = new List<Drank>
{
    new Drank("Koffie", 2.5F),
    new Drank("Thee", 3),
    new Drank("Kleine soep", 4),
    new Drank("Grote soep", 6)
};

Dranken.RemoveAll(d => d.Prijs > 3);

foreach (Drank d in Dranken)
{
    Console.WriteLine(d.Naam);
}

/*
Lijst dranken is nu veranderd
output:
Koffie
Thee
*/
```

Andere methodes die op een gelijkaardige manier werken zijn **Contains**, **Exists** en **TrueForAll**. Al de properties en methodes vind je hier terug:

https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-5.0

ToString()

Tot nu toe hebben we altijd een eigen methode verzonnen om een object te tonen (PrintAuto, ToonPersoon, …). Maar elk object dat je aanmaakt heeft eigenlijk al een eigen methode omzichzelf te tonen: ToString. Alleen zal je zien dat als je deze methode aanroept je gewoon de naam van de class terugkrijgt als string. Dat is niet erg nuttig.

De ToString() methode wordt daarom vaak vervangen door een eigen versie. Om dit goed te laten gaan moet je het woordje **override** toevoegen voor het return-type. Een voorbeeld in een mogelijke class Persoon (met velden _voornaam, _achternaam en _woonplaats).

```
public override string ToString()
{
    return $"{_voornaam} {_achternaam} woont in {_woonplaats}.";
}
```

Het handige aan de ToString methode is dat deze automatisch zal aangeroepen worden als je bijvoorbeeld een object gaat WriteLinen (je hoeft dus niet nog eens .ToString() achter het object te zetten).

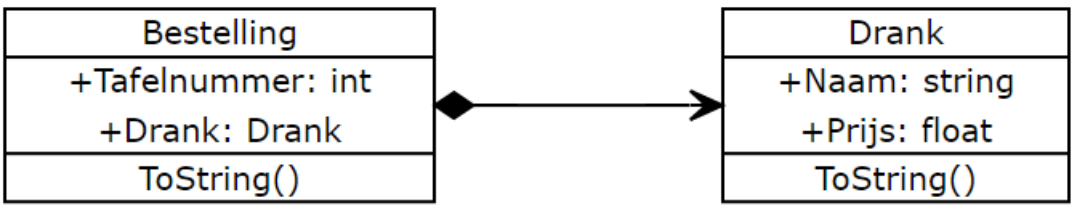
```
Persoon p1 = new Persoon("Jos", "Vermeulen", "Diepenbeek");

Console.WriteLine(p1); // .ToString() hoef je niet toe te voegen
```

Oefening1: Refactoring Café Sport

Herschrijf het progamma Café Sport met classes.

Voorzie een class Bestelling en een class Drank.

Een Bestelling object heeft een tafelnummer en een drank.



Voorzie een lijst waar de Bestellingen in worden bijgehouden en een lijst waar al de verschillende Dranken in komen te staan. (Als er bijvoorbeeld een tafel meerdere koffie's wil

bestellen worden er meerdere keren een gelijkaardig bestelling object in de lijst met bestellingen gezet.)

Oefening 2: Café Sport met aantal

Omdat een tafel meerdere dezelfde dranken kan bestellen is het logisch van per bestelling ook een aantal bij te houden. Herschrijf je programma zodat een bestelling een tafelnummer, een drank en een aantal heeft.

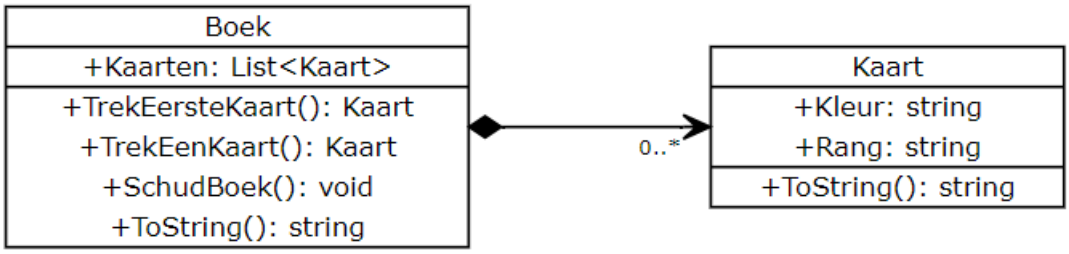
Bij het bestellen van 3 koffie's zou er in de bestellingen-lijst 1 object moeten zitten waarvan het aantal op 3 staat.

Oefening 3: Kaartspel

Maak een class "Kaart" met een eigenschap "Kleur" (voor de kaartsymbolen ruiten, klaveren, harten, schoppen) en een eigenschap "Rang" (voor Aas, 2, 3, 4, …, Koning).

Maak een class "Boek" waarin je een lijst van Kaart-objecten in kan bewaren. Voorzie in het Boek-object methodes om kaarten op te vragen zoals

- TrekEersteKaart (eerste kaart uit de lijst met kaarten wordt gegeven)
- TrekEenKaart (een willekeurige kaart uit de lijst wordt gegeven)
- SchudBoek (de lijst wordt dooreen gezet)



Oefening 4: Hoger/lager

Kan je een spelletje maken waarbij de gebruiker een kaart gepresenteerd krijgt en moet raden of de volgende kaart hoger of lager zal zijn? Als de kaart getrokken is herhaalt het spel zich met de nieuw getrokken kaart tot alle 52 kaarten getrokken zijn. Hou ook een score bij.