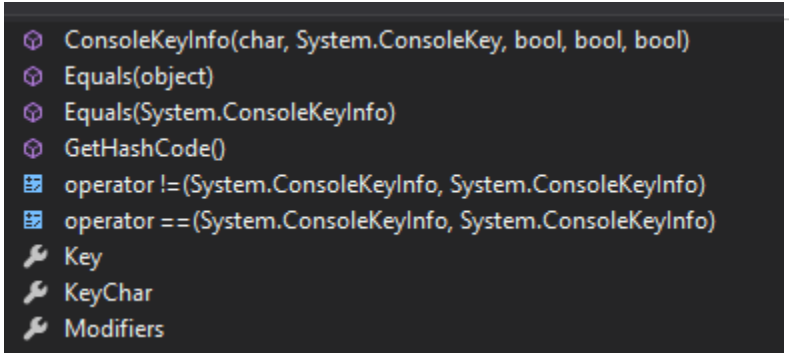
Programmeertechnieken & Ontwerpen

Objecten

Class vs object

```
1  ConsoleKeyInfo consoleKeyInfo = Console.ReadKey();
```

Class Object



Een **klasse** (class) is een **blueprint** van iets. Een klasse bevat dus **geen concrete informatie**. Wel **bepaald** een klasse **welke informatie** we kunnen verkrijgen.

Een **object** daarintegen is een **specifieke invulling van een klasse**. Een object bevat **concrete info.**

Using & namespaces

Om structuur in de klasses te krijgen en hergebruik van klassenamen mogelijk te maken. Maken we gebruik van namespaces.

Een **namespace** is een **soort groep waartoe een bepaalde klasse behoord.**

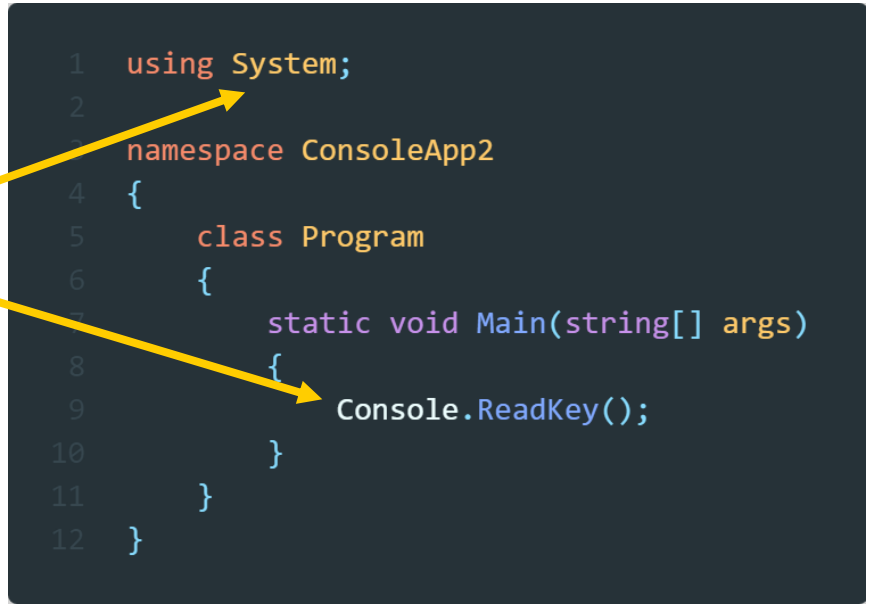
Zelf hebben we reeds onbewust al gebruik gemaakt van namespaces. Telkens wanneer we een nieuwe applicatie gemaakt hebben, hebben we ook voor elk project een nieuwe namespaces gemaakt.

```
1  namespace ConsoleApp2
2  {
3      class Program
4      {
5          static void Main(string[] args)
6          {
7          }
8      }
9  }
```

Using & namespaces

Telkens wanneer we een klasse gaan gebruiken, moeten we er dus voor zorgen dat onze compiler weet waar hij deze klasse moet halen. Dit geven we aan a.d.h.v **using**.

De class **Console** kunnen we terug vinden in de namespace **System**. We moeten deze dus toevoegen.



```
1  using System;
2
3  namespace ConsoleApp2
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.ReadKey();
10         }
11     }
12 }
```

Using & namespaces

Standaard wanneer we een nieuw project maken zijn er heel wat usings toegevoegd.
Sommige hiervan zijn overbodig. Door gebruik te maken van onze code cleanup tool worden deze automatisch verwijderd. Dit zorgt ervoor dat ons programma sneller compileert aangezien we minder externe bronnen nodig hebben.

Let wel op het kan gebeuren dat je hierdoor bepaalde klassen niet direct terug vind.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp3
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

The 3 parts of a class

Een klasse bestaat typisch gezien uit 3 belangrijke delen.

1. Constructor
   Een soort “method” die er voor zorgt dat er een object gecreëerd wordt van een klasse.
2. Methodes
   Deze voeren iets uit dat betrekking heeft tot het object.
3. Properties
   Eigenschappen van een object.

Constructor

De constructor hebben we nodig om **object** te **initialiseren**.

```
1  Random random = new Random();
2  DateTime dateTime = new DateTime(2019,12,13);
3  int number = 0;
```

In bovenstaand voorbeeld zijn *Random()* en *DateTime(2019, 12 ,13)* de constructors. Ze creëren een specifiek object van het type Random en DateTime.

Voor simpele (value) datatypes hebben we geen constructor nodig (int). Complexere types die gebaseerd zijn op een klasses hebben we wel een constructor nodig (Random, DateTime)

Object declareren

```
1   Random random;
```

Net zoals bij onze value types bestaat de object declaratie uit een type (class) en een variabele naam (object).

Het grote verschil hierbij is dat **value types** (int, double, long, ...) een **default waarde** hebben wanneer ze nog niet geïnitialiseerd zijn.

Objecten die niet geïnitialiseerd krijgen een NULL value. Wanneer we hier niet mee opletten resulteert dit in een NullRefereceException.

Initialisatie

```
1  Random random = new Random();
2  DateTime dateTime = new DateTime(2019,12,13);
```

Object initialisatie gebeurd op een andere manier dan met value types. We kunnen bovenstaande lijnen als volgt lezen:

Maak een object random aan van de klasse Random en ken daar een nieuw object Random aan toe.

Maak een object dateTime aan van de klasse DateTime en ken daar een nieuw object DateTime aan toe met enkele start properties (jaar, maand, dag).

De initialisatie is dus een aanroep van de constructor met het keyword **new** voor. De constructor heeft steeds dezelfde naam als de klasse.

Methods vs properties

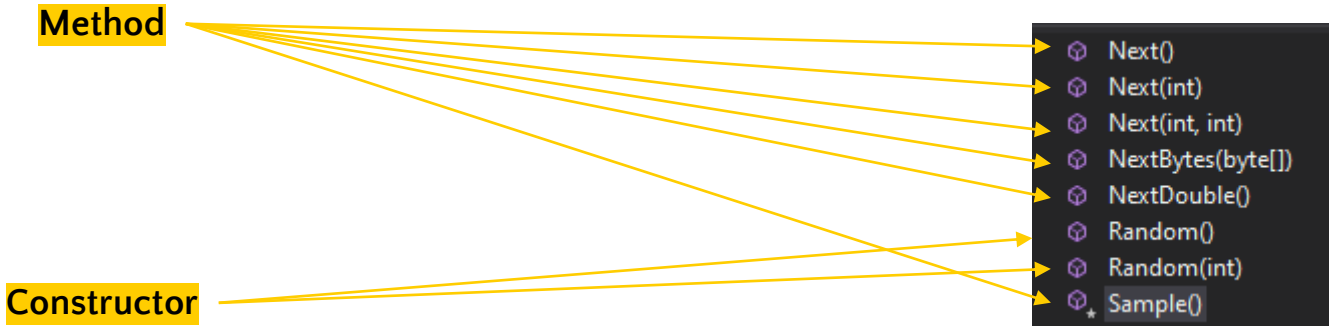
Zoals reeds eerder gezegd een **methode** zal iets doen dat betrekking heeft op het object. *ReadKey()* is een methode die iets inleest. De klasse Console handelt dit achterliggend voor ons af.
Een methode bevat altijd **2 haakjes** met al dan niet enkele **argumenten** en kan **iets terug geven**.

Een **property** bevat een eigenschap van een object. KeyChar is een property van het ConsoleKeyInfo object.
Een **property** is het best te vergelijken met een variabele. We kunnen deze **uitlezen** of er iets naar **wegschrijven**.

```
1  ConsoleKeyInfo consoleKeyInfo;
2  char pressedChar;
3  consoleKeyInfo = Console.ReadKey(); //methode
4  pressedChar = consoleKeyInfo.KeyChar; // property
```

Random

Een klasse die we nu kunnen gebruiken is bijvoorbeeld de klasse random.



Het hebben van meerdere constructors noemen we net zoals bij methods overloading. Volledig heet dit **constructor overloading.**

DateTime

DateTime is ook een heel handige klasse. Helaas bestaat deze klasse uit teveel constructoren, methodes en properties om hier volledig te printen.

Meer info over deze klasse kan hier gevonden worden:
https://docs.microsoft.com/en-us/dotnet/api/system.datetime?view=netframework-4.8#constructors

Of in visual studio in de object browser.

Boek

Behandelde hoofdstukken

- H.6 Objecten
  - 6.1, 6.2, 6.3, 6.5, 6.6, 6.7