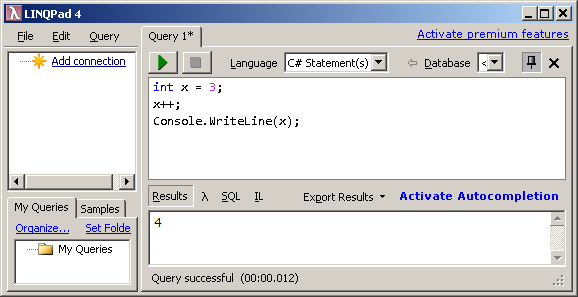
# Werkcollegebundel .NET Programmeren 2011-2012

auteur: ir. G.M. Borkent

De opgaven in deze bundel zijn bedoeld om te maken in de les. Ze zijn genummer x.y, x staat hierbij voor het weeknummer.

# LinqPad

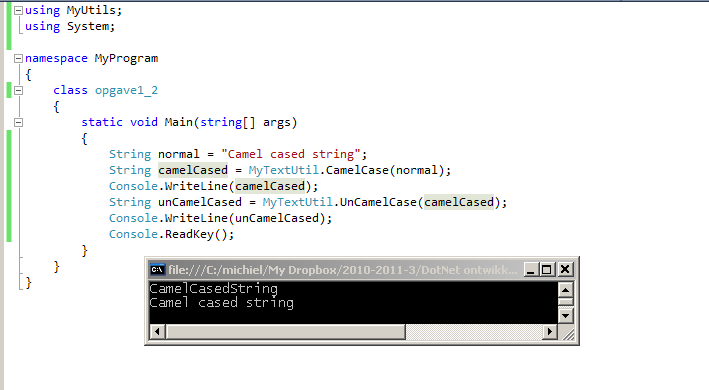
In de werkcollege- en practicumopdrachten wordt er in principe vanuit gegaan dat er Visual Studio 2010 Professional wordt gebruikt. Echter, soms wil je even snel een klein stukje C# uitproberen, zonder daarvoor een nieuwe Solution en/of Project te hoeven aanmaken in Visual Studio. Voor dat doel kun je LinqPad gebruiken. Ga daarvoor naar: <http://www.linqpad.net/> en download de versie voor .NET 4.0. Deze versie is gratis, maar bevat helaas geen AutoCompletion, zoals Visual Studio (wel kan je deze feature tegen betaling toevoegen).



# Opdracht 1.1

Maak een class met twee static methoden CamelCase en UnCamelCase.

De methoden moeten als volgt zijn aan te roepen:



Bekijk verder de methoden van System.Char en System.String, deze zullen je handig van pas komen. Reproduceer de output die je hierboven ziet. Het moet uiteraard voor andere strings ook werken.

# Opdracht 1.2 (properties)

Schrijf een:

* class Product met:
  + een Name property (lezen en schrijven)
  + Property PriceInDollars (alleen lezen)
  + Property PriceInEuros (alleen lezen)
  + gebruik als omrekenfactor dollar -> euro 0,75
* een Main methode met:
  + twee objecten van class Product die je verzamelt in een array
  + laat de naam en beide prijzen van beide producten zien middels een foreach loop, zodat de volgende output verschijnt:



# Opdracht 1.3 (properties, read only fields)

Bestudeer onderstaand programma en be-argumenteer wat er goed en/of fout is aan:

1. de property Created
2. het readonly attribuut created

mmmmmmmmm

using System;

using System.Threading;

namespace opgave1\_3

{

class opgave1\_3

{

static void Main(string[] args)

{

MyObject o1 = new MyObject();

Thread.Sleep(1000);

MyObject o2 = new MyObject();

Console.WriteLine("o1 was created: {0} and o2 was created: {1}",

o1.Created, o2.Created);

Console.WriteLine("o1 was created: {0} and o2 was created: {1}",

o1.created, o2.created);

Console.ReadKey();

}

}

class MyObject

{

public MyObject()

{

\_created = DateTime.Now;

created = DateTime.Now;

}

private DateTime \_created;

public readonly DateTime created;

public DateTime Created

{

get

{

return \_created;

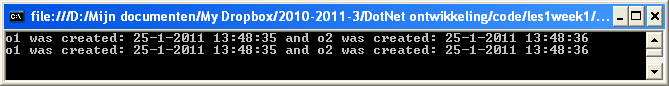
}

}

}

}

Mogelijk uitvoer van het programma:



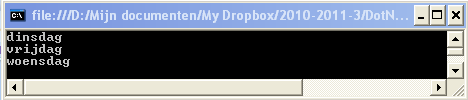
# Opdracht 1.4 (enums)

Een enum is een verzameling van gerelateerde constante waarden die je een naam kan geven, zoals de dagen van de week, de maanden van het jaar, burgerlijke status, leeftijdscategorie, etc.

Schrijf een enum genaamd Days die de werkdagen van de week bevat (maandag tot en met vrijdag). Omdat het ‘governing type’ van een enum een built-in integer type is, zoals byte, sbyte, short, ushort, uint of long, kan je een enum ook gebruiken in een switch-statement.

Creeër twee of meer verschillende dagen van type Days en gebruik een switch statement om een string-representatie van deze dagen naar de Console weg te schrijven.

Mogelijke output:



# Opdracht 1.5 (ValueTypes en ReferenceTypes)

1. In C# geldt: ‘everything is an object’. Alle objecten erven uiteindelijk van de class Object. Dit geldt ook voor Value Types. Zoek uit hoe de class hierarchy van een Value Type, zoals System.Single eruit ziet.
2. Pas het onderstaande LinqPad-voorbeeld zo aan dat:

1 de foutmelding bij het compileren verdwijnt

2 de output van het programma ‘changed’ wordt

struct MyStruct

{

public string StructName { get; set; }

public MyStruct(string name)

{

StructName = name;

}

}

void Main()

{

MyStruct m = new MyStruct("name");

ChangeStruct(m);

Console.WriteLine(m.StructName);

}

void ChangeStruct(MyStruct m)

{

m.StructName = "changed";

}

# Opdracht 2.1 (Inheritance/polymorfisme)

Gegeven is de volgende Java-code en bijbehorend UML-diagram.

**public** **static** **void** main(String[] args) {

Koe k = **new** Koe();

Eend e = **new** Eend();

Dier[] ds = {k,e};

**for** (Dier d : ds) {

System.*out*.print(d);

System.*out*.println(" en ik zeg " + d.speak());

}

e.setKanVliegen(**false**);

k.setLitersMelk(6);

**for** (Dier d : ds) {

System.*out*.print(d);

System.*out*.println(" en ik zeg " + d.speak());

}

}

public abstract class Dier {

protected int aantalPoten;

public Dier() {

aantalPoten = 0;

}

public Dier(int aP) {

aantalPoten = aP;

}

public abstract String speak();

public String toString() {

return "Ik heb " + aantalPoten + " poten";

}

}

public class Koe extends Dier {

private double litersMelk;

public Koe() {

super(4);

}

public void setLitersMelk(double lt) {

litersMelk = lt;

}

public String speak() {

return "Boe!";

}

public String toString() {

return "Ik ben een koe met " + litersMelk + " liters melk. " + super.toString();

}

}

public class Eend extends Dier {

private boolean kanVliegen;

public Eend() {

super(2);

kanVliegen = true;

}

public void setKanVliegen(boolean b) {

kanVliegen = b;

}

public String speak() {

return "Kwek!";

}

public String toString() {

return "Ik ben een eend en ik kan " + (kanVliegen ? "wel" : "niet") + " vliegen. "+ super.toString();

}

}

UML-diagram:



1. Wat betekent het dat de klasse Dier abstract is?
2. Neem de Java-code over in een Linqpad-query bestand. Je kunt de code uit de methode main over nemen in de Main-methode die aangemaakt wordt bij een “C# Program” in Linqpad. De overige classes kun je eronder kopiëren.
3. Run het programma.
4. Welke foutmelding(en) zie je? Schrijf ze op.
5. Wijzig de code om de huidige foutmelding te verhelpen en herhaal stap c, d en e totdat de juist output verschijnt die hieronder is gegeven. **Laat hierbij de class Dier** ongewijzigd!

Ik ben een koe met 0.0 liters melk. Ik heb 4 poten en ik zeg Boe!

Ik ben een eend en ik kan wel vliegen. Ik heb 2 poten en ik zeg Kwek!

Ik ben een koe met 6.0 liters melk. Ik heb 4 poten en ik zeg Boe!

Ik ben een eend en ik kan niet vliegen. Ik heb 2 poten en ik zeg Kwek!

# Opdracht 2.2 (Inheritance/polymorfisme)

Deze opgave bouwt voort op opgave 2.1.

In de vorige opgave mocht je de klasse Dier niet aanpassen. Nu gaan we dat wel doen.

1. Verander in klasse Dier toString in ToString (met een hoofdletter T dus) en compileer nogmaals. Als je het goed is krijg je nu de foutmelding:

Koe.ToString()': cannot override inherited member 'UserQuery.Dier.ToString()' because it is not marked virtual, abstract, or override

1. Leg uit wat deze foutmelding betekent. Voeg de modifier virtual toe en compileer nogmaals. Als je de laatste beta-versie van Linqpad hebt, krijg je nu de warning dat de methode ToString van Dier de methode ToString van object ‘verstopt’ (method hiding). Wat betekent deze foutmelding? Zoek in het boek op blz 109-111 op wat ‘method hiding’ is.
2. Bekijk de output van vraag b en bedenk welke ToString-methode er wordt aangeroepen door Console.WriteLine. Verander nu de modifier van methode ToString in class Dier in override. Welke methode van welke class wordt er nu dus ‘ge-override’?

# Opdracht 2.3 (Delegates)

1. Schrijf een C#-programma in Linqpad dat twee methoden bevat:
   * De ene methode telt twee ints bij elkaar op en geeft het resultaat terug
   * De andere methode vermenigvuldigt twee ints en geeft het resultaat terug
2. Definieer een delegate die dezelfde signatuur heeft als bovenstaande methoden
3. Maak in de methode Main een lijst van het delegate-type gedefinieerd in vraag b die verwijzingen naar de methoden uit vraag a bevat.
4. Roep middels een foreach-loop de methoden uit de lijst van vraag c aan met de getallen 5 en 6. Print de uitkomst van elke methode op het scherm. Welke uitvoer zie je?

# Opdracht 2.4 (Anonieme methoden)

Neem de uitwerking van de vorige opgave over. Verwijder de twee methoden die je bij a) gemaakt hebt en vervang ze door anonieme methoden.

# Opdracht 2.5 (lambda functies)

1. Maak twee functies met behulp van de Func delegate en lambda-functies genaamd add en mul die respectievelijk twee ints kunnen optellen en vermenigvuldigen.
2. Wat is de waarde van a in

int a = mul(add(4, 5), 10);

Controleer de uitkomst.

# Opdracht 3.1

Schrijf een methode TelOp waarmee je een willekeurig aantal ints op kan tellen.

De methode moet dus als volgt aan te roepen zijn:

int x = TelOp();

int y = TelOp(1);

int z = TelOp(1, 2);

int u = TelOp(10, 11, 12);

int som = TelOp(x, y, z, u);

Controleer in je programma of de waarde van som is wat je verwacht.

# Opdracht 3.2

De onderstaande string bevat een tekst die in de vormgevingsindustrie vaak gebruikt wordt als dummy tekst.

String lorem = "Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum";

1. Sla elk woord afzonderlijk op in een array van strings, zodat de array alle woorden van de tekst bevat in de juiste volgorde. Gebruik daarvoor de methode Split van de class String (zie ook blz 225 in het boek).

Gebruik lambda expressies om aan het antwoord op de volgende vragen te komen.

1. Zoek het eerste woord dat met een ‘a’ begint. Druk het woord af op het scherm.
2. Zoek alle woorden die met een ‘a’ beginnen en sla ze op in een array. Druk ze af op het scherm. Laat tevens zien hoeveel het er zijn.
3. Hoeveel woorden zijn korter dan 3 karakters? Druk dit aantal af op het scherm.

# Opdracht 3.3

1. Schrijf een class Product met property Price van type double. De class moet de IComparable interface implementeren.
2. Maak een niet gesorteerde array aan van objecten van bovenstaande class.
3. Sorteer de array van laag naar hoog.
4. Sorteer de array van hoog naar laag middels een lambda expressie.

# Opdracht 3.4

Bij deze opdracht is het aan te raden om Visual Studio te gebruiken ivm het automatisch aanvullen bij het implementeren van IEnumerable.

Met met behulp van de interface IEnumerable en het yield return statement is het mogelijk om een oneindige lijst te maken. Zo kan je bijvoorbeeld een lijst maken waarvan het volgende element de waarde van het vorige element maal 2 is: { 2, 4,8,16,32, …}.

Maak een class Reeks die de interface IEnumerable implementeert en deze reeks oplevert.

Maak een object van deze class aan en vraag middels een foreach loop, dus foreach (int i in r) ... , waarbij r van het type Reeks is, de opeenvolgende waarden uit en print ze op het scherm, totdat de waarde groter is dan 1000.

# Opdracht 3.5

Pas de class uit de vorige opgave aan. Schrijf nu een class RekenReeks. Deze class heeft een op te geven beginwaarde (int) en een functie die op basis van de vorige waarde de volgende waarde kan berekenen. Zo levert:

RekenReeks r = new RekenReeks(2, n => n \* 2);

de reeks uit vraag 3.4 op.

# Opdracht 4.1

Gebruik bij deze opgave de class Reeks uit opdracht 3.4. Deze leverde de lijst met machten van 2: {2,4,8,16, 32, ...}.

1. Declareer een LINQ-expressie de getallen uit deze reeks ophaalt die groter zijn dan 10 en kleiner dan 1000.
2. Welk gevaar bestaat er bij het rechtstreeks uitvragen van alle elementen uit bovenstaande LINQ-expressie?
3. Bekijk de extension-methods SkipWhile en TakeWhile uit System.Linq.Enumerable en haal met behulp van deze methoden de bij a gevraagde waarden op. Druk ze af op het scherm.

# Opdracht 4.2

Schrijf een extension-method TakeRange voor de class Reeks uit opgave 3.4. De extension-method moet twee ints als argument meekrijgen, welke de onder- en bovengrens van de selectie voorstellen.

Zo kun je bijvoorbeeld de bij opgave 4.1 gevraagde range opvragen:

Reeks r = new Reeks();

IEnumerable<int> selectie = r.TakeRange(10, 1000);

Je mag de class Reeks zelf niet aanpassen. Sla er acht op dat extension-methods gedefinieerd moeten zijn in static classes.