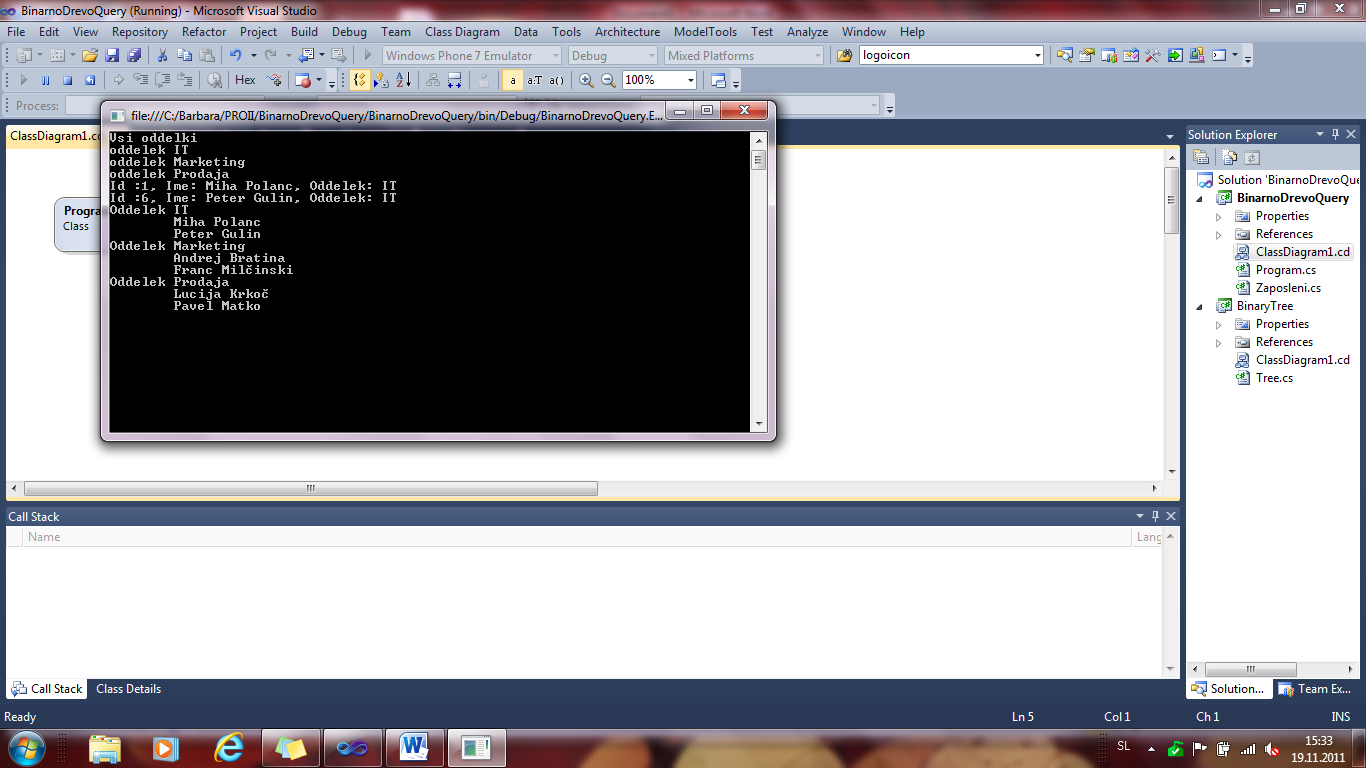
# LINQ poizvedbe v drevesu

V nalogi bomo uporabljali LINQ sintakso za dostop do poljubne zbirke podatkov, ki implementira vmesnik IEnumerable. V našem primeru bomo uporabili napisan projekt BinaryTree (v spletni učilnici), ki nam podatke v programu razvrsti v binarno drevo.

Izdelajmo si novo konzolno aplikacijo in vanjo dodajmo razred Zaposleni. Razred naj ima spodnji UML diagram



Metoda ToString naj izpiše vse podatke o zaposlenem spodnji obliki



Metoda CompareTo je del vmesnika IComparable, ki zahteva naš generičen razred BinaryTree (ki ga bomo dodali projektu). Metoda CompareTo naj vrne 1,-1 in 0 glede na Id zaposlenega.

Sedaj pa v projekt dodajmo drugi projekt, BinaryTree. V Solution Explorerju izberemo našo rešitev (pazite, da ne boste izbrali samo projekta, ampak celotno rešitev) in uporabimo možnost Add, nato pa Existing Project. Dodamo projekt BinaryTree. Projekta ne bomo popravljali, ampak ga bomo samo uporabili. UML diagram razreda Tree je



Uporabljali bomo konstruktor razreda in metodo Insert, s pomočjo katere vstavimo nov zapis v drevo.

Vrnimo se k našemu projektu. Najprej pod Reference dodamo projekt BinaryTree, nato pa v razredu Program.cs dodamo

using BinaryTree;

static void Main(string[] args)

{

try

{

Delaj();

Console.ReadLine();

}

catch (Exception ex)

{

Console.WriteLine("Exception: {0}", ex.Message);

}

}

static void Delaj()

{}

Vso kodo bomo sedaj pisali v metodo Delaj. Najprej si napolnimo drevo z nekaj zaposlenimi. Na primer

Tree<Zaposleni> z = new Tree<Zaposleni>(

new Zaposleni { Id = 1, Ime = "Miha", Priimek = "Polanc", Oddelek = "IT" });

z.Insert(new Zaposleni {Id = 2, Ime = "Andrej", Priimek = "Bratina", Oddelek = "Marketing"});

z.Insert(new Zaposleni { Id = 4, Ime = "Lucija", Priimek = "Krkoč", Oddelek = "Prodaja" });

z.Insert(new Zaposleni { Id = 6, Ime = "Peter", Priimek = "Gulin", Oddelek = "IT" });

z.Insert(new Zaposleni { Id = 3, Ime = "Franc", Priimek = "Milčinski", Oddelek = "Marketing" });

z.Insert(new Zaposleni { Id = 5, Ime = "Pavel", Priimek = "Matko", Oddelek = "Prodaja" });

Sedaj pa rešite spodnje naloge

//1. napiši kodo, ki izpiše vse oddelke

//2. napiši kodo, ki izpiše vse različne oddelke

//3. napiši kodo, ki izpiše vse zaposlene v oddelku IT

//4. napiši kodo, ki izpiše zaposlene po oddelkih

LINQ uporablja t.i. odloženo izvajanje. Poizvedba se izvede šele, ko program potrebuje rezultate. Zakomentirajte v metodi Delaj() svoje rešitve in prepišite spodnji del kode:

Console.WriteLine("Vsi zaposelni");

var a = from b in z

select b;

foreach (var k in a)

Console.WriteLine(k.ToString());

z.Insert(new Zaposleni { Id = 7, Ime = "Monika", Priimek = "Seleš", Oddelek = "Kozmetika" });

Console.WriteLine("Vsi zaposelni");

foreach (var k in a)

Console.WriteLine(k.ToString());

Najprej izberem in izpišem vse zaposlene, nato dodam enega zaposlenega, nato pa ponovno izpišem vse zaposelne. Kaj opazite?

Če ne želite zgoraj prikazanega obnašanja, morate zahtevati, da se poizvedba izvede takoj. To naredite s spremembo poizvedebe

var a = from b in z.ToList<Zaposleni>()

select b;

Rešitev:

//1. napiši kodo, ki izpiše vse oddelke

//2. napiši kodo, ki izpiše vse različne oddelke

Console.WriteLine("Vsi oddelki");

var o = (from a in z

select a.Oddelek).Distinct();

foreach (var k in o)

Console.WriteLine("oddelek " + k);

//3. napiši kodo, ki izpiše vse zaposlene v oddelku IT

var o1 = from a in z

where a.Oddelek == "IT"

select a;

foreach (var k in o1)

Console.WriteLine(k.ToString());

//4. napiši kodo, ki izpiše zaposlene po oddelkih

var o2 = from a in z

group a by a.Oddelek;

foreach (var k in o2)

{

Console.WriteLine("Oddelek " + k.Key);

foreach (var k1 in k)

Console.WriteLine("\t{0} {1}", k1.Ime, k1.Priimek);

}