# C# 3.0

## Labs

# Lab 1

## Exercises

1. Create a Visual Studio 2010 solution and add a Console project.
2. Create a partial class named BankAccount.
3. Add three auto-implemented properties: uint AccountNumber, decimal Credit and string Owner.
4. Add a partial method to the BankAccount class named: AnonymiseName. Call this method in the get of the Owner property.
5. Create a List of BankAccount and add several instances to the list using the (collection) Initializers.
6. Write all accounts to the console using a foreach loop and var for the type of the elements.
7. Add another partial class BankAccount. Add the partial AnonymiseName method. In this method: replace the owner field of the bank account by a string of the same size but containing all dashes. Run the application to see the effect.
8. Call the FindAll method on the list of BankAccount and pass a delegate to a function that returns true when the credit of a given BankAccount is below a specific level.
9. Call FindAll again but now by using a Lambda Expression.

# Lab 2

## Setup

Open the starter Visual Studio solution.

Examine the DataHelper project. This project contains a DataSource class that provides access to four collections of objects: Employees, Products, ProductVendors and Vendors.

Examine the Program class in the Lab\_02 project.

For each of the following exercises you should add a method to the Program class just like the Exercise01 method that is already present in the starter and add code to the method.

Do not forget to call the added method from within the Main method.

## Exercise 1

Use the extension methods of the Enumerable class to answer these questions:

1. Are there any employees with less than 21 sick leave hours?
2. How many employees have less than 21 sick leave hours?
3. For each employee that has less than 21 sick leave hours, display his/her name, gender and number of sick leave hours.
4. Display the same list but now double sorted, first by gender and then by name.

## Exercise 2

1. Rewrite the queries from Exercise 2 using the Comprehension Syntax.

## Exercise 3

For the Products, ProductVendors and Vendors collections:

1. Display for each product in the Products collection the name of the product and the prices it is being offered for.
2. Add to the result of the previous question the name of the Vendor.
3. Using the result of the previous question, group the result by product and display the offerings in groups like this:  
     
   Product: LL Mountain Rim  
   Offering: 22,28 by Comfort Road Bicycles  
   Offering: 21,11 by Competition Bike Training Systems  
     
   Product: ML Mountain Rim  
   Offering: 26,33 by Comfort Road Bicycles  
   Offering: 27,89 by Competition Bike Training Systems  
     
   Product: HL Road Pedal  
   Offering: 59,99 by Compete Enterprises, Inc  
   ...
4. Display for each product: the name of the product, the price and the name of the vendor of the cheapest offering for that product. When there is more than one vendor offering the lowest price show all vendors that offer the lowest price.

## Exercise 4

Create a method named Exercise04a. Create a LINQ expression that selects from all employees: the name and his/her sick leave age ratio. The sick leave age ratio is defined like this:

SickLeaveAgeRatio = employee.SickLeaveHours \* 1000 / (DateTime.Now - employee.BirthDate).TotalHours

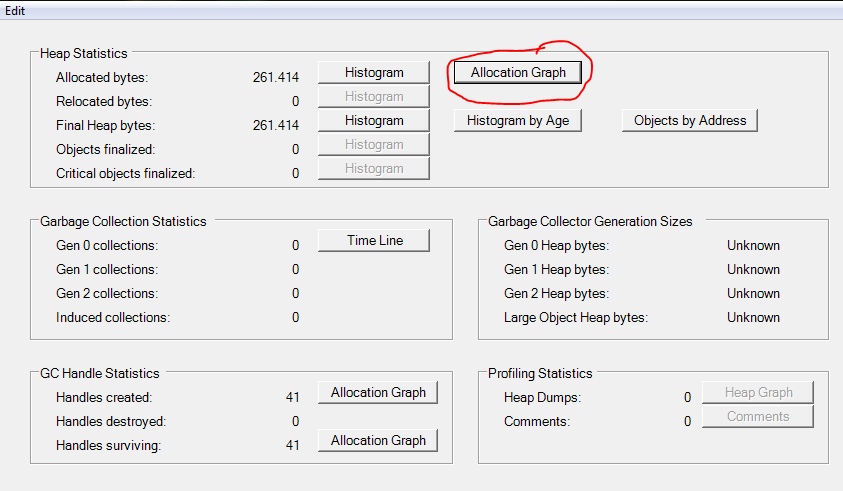
Order the collection by the sick leave age ratio.

Create an instance of the system.Diagnostics.Stopwatch class and start the clock. Force the evaluation of the LINQ expression by calling ToArray on the collection. Store the ElapsedMilliseconds value of stopwatch immediately after the ToArray call in a variable. Stop the stopwatch and write to the console how many milliseconds it took for the execution of ToArray.

Copy the Exercise04a method to a new Exercise04b method and add a *let-clause* that calculates the sick leave age ratio and stores it in a range variable. Use this range variable in the orderby and select clauses.

The method Exercise04 should call both Exercise04a and Exercise04b. Make sure only the method Exercise04 is called in the Main method. Run the application twice to see the timing results. The first time forces the Just-In-Time compilation, therefore the second run gives more valid readings. Note which version is faster.

Start the CLRProfiler and run the application using the CLRProfiler. Let the application finish so the Summary window of the CLR Profiler opens.



Click the Allocation Graph button in the Heap Statistics group.

Set the Detail level to 1 and look for the statistics on the Exercise04a and Exercise04b methods. Note the memory use of both.

Open the assembly using Reflector and inspect the Exercise04a and Exercise04b methods. Use View | Options, Optimization set to .NET 2.0 to force translation from comprehensive syntax to method calls.

Can you explain the differences in execution time and memory use of both methods?