**INT422 – Lecture 3-2**

Pattern wrap-up, get all, get one, add new, edit existing, delete item. Work with data annotations, and LINQ. Begin work on Assignment 3.

**Code examples reminder**

This week, we have three code examples:

* EditDelete
* DataAnnotations
* LINQIntroduction

Please remember to include them in your studying process. Open them in Visual Studio, and run them. Where appropriate, re-create them, or parts of them, on your own.

**Pattern wrap-up and summary**

In the past two weeks, you have learned how to handle common interaction patterns – with a single data entity (object or collection) – by using best practice design and coding techniques.

The short-term goal was to introduce work habits that are repeatable, reliable, and efficient.

The medium-term goal is to prepare you to work with associated entities, which is a more real-world scenario.

The longer-term goal is to enable you to design and develop a great web app, which has some complexity, and can scale. Or a mobile app, because the patterns are similar.

Your teachers have prepared a fill-in worksheet that you can use to record these patterns, and then revise and use as a quick reference in the future. (It’s a Word document.) In class, we’ll get started on the exercise.

[Patterns summary – single entity](https://scs.senecac.on.ca/~wei.song/int422/common/patterns-summary-single-entity.docx" \t "_blank)

You’re required complete worksheet as a homework (2%) and submit it to the blackboard before the end of Saturday of week 4. Print out a hard copy for your own use as a quick reference in the future."

**Popular LINQ features**

As you have learned, we will most often use LINQ for query tasks:

* locating/selecting one item (e.g. “find”, “single or default”)
* filtering a collection to select items which match a condition (e.g. “where”)
* sorting (ordering) a collection (e.g. “order by”)

Let’s look at all of these, in a bit more detail.

**Fetching one single matching object**

In the past, you have used the [Find()](https://msdn.microsoft.com/en-us/library/gg696418%28v=vs.113%29.aspx) method.

This method works only when the following two conditions are true:

1. You are working with a **DbSet<TEntity>**
2. You are working with the **primary key** property (the one named “Id” or “<entity-name>Id”)

Now, consider a situation where we have a Seneca Student entity. Each student has (at least) two properties which will have unique values: Seneca Student Identification Number (Student ID), and Canada Social Insurance Number (SIN). Neither property is the primary key.

Can you use Find() to locate an object by Student ID or SIN?

No.

Why? Neither property is the primary key.

How can we locate the object? With the [SingleOrDefault()](https://msdn.microsoft.com/en-us/library/bb549274%28v=vs.110%29.aspx) method. It requires a **lambda expression** as an argument. Here’s how we can use this method:

// The Find() method, when we can use the primary key to locate the object

var supplier = ds.Suppliers.Find(newItem.SupplierId);

// The SingleOrDefault() method, when we need to use a lambda expression

var student = ds.Students.SingleOrDefault(s => s.SIN == newItem.SIN);

❝**More about SingleOrDefault() and Find()**

There are several extension methods that a beginner may view as similar:

* Find
* First
* FirstOrDefault
* Single
* SingleOrDefault

We will use only two:

* As noted above, if we are working with the primary key of a DbSet<TEntity>, we can use **Find**
* Otherwise, we must use **SingleOrDefault**

Why?

A successful SingleOrDefault expects to return exactly one object.

An unsuccessful SingleOrDefault will return **null**.

This is the behaviour we want, when we are working with a (primary key) identifier. In our code, we simply need to do one check/test – “if null” – to determine how to use the result.

In contrast, FirstOrDefault is used when the data source contains zero or more matching objects (that meet the fetch condition). If successful, it returns the first object found. If unsuccessful, null is returned. As a result, you would have to perform two checks/tests – number of objects, and null state.

First or Single should not be used, when following our coding standards. If First or Single is unsuccessful, the statement it appears in will raise an exception. In other words, it will not assign null as the return value, as with the related methods FirstOrDefault or SingleOrDefault.

.

**Filtering**

Within a manager method (and before returning the results), we often need to perform other tasks on a collection, such as filtering or sorting.

For example, assume that we wish to select only those Program objects that have a Credential property value of “Diploma”:

var diplomaPrograms = ds.Programs.Where(p => p.Credential == "Diploma");

❝Reminder: “p” is the range variable.

It represents an object in the Programs collection. Its type is Program.

The letter “p” does not have any other special significance.

Like other variables, its name is meaningful to you only. Choose your own name. It can be one or more characters in length. Maybe use the first character or two of the type in the collection.

.

**Sorting**

In this example, assume that we wish to sort the Program collection by a property named Code:

var sortedPrograms = ds.Programs.OrderBy(p => p.Code);

The [OrderBy](http://msdn.microsoft.com/en-us/library/bb534863.aspx) extension method is defined in the Queryable class. This method can be used on any object of type IQueryable<TEntity>. Its return type is [IOrderedQueryable<T>](http://msdn.microsoft.com/en-us/library/bb340178.aspx).

If you need to sort on a second (or third) property, use additional ThenBy() methods, which use the same style of lambda expression.

Also, each has a variant which will sort in descending order: OrderByDescending() and ThenByDescending().

.

**Filtering and sorting**

You can combine tasks by chaining the filtering and sorting tasks above (using the “fluent” syntax). Do the filter task first, and then do the sort task. The statement below has been wrapped for clarity:

var c = ds.Programs

    .Where(p => p.Credential == "Degree"))

    .OrderBy(p => p.Code);

Here’s how to read this statement, in English:

Fetch the Program collection…  
but only where a Program object’s Credential property value…  
matches the string “Degree”…  
then sort the result by the Program object’s Code property value.

**Return types**

This is a very important topic.

In many Manager class methods, you will want to return the result of a query. What return type must be used?

Answer: Usually, a type that’s based on a view model class.

❝Again, this is very important.

Earlier, you learned that the Manager class must accept and deliver types that are either 1) based on the .NET Framework types, or 2) your own custom types, including those defined in view model classes.

Do NOT return results directly from the data context. Never.

What can we return?

For a single object, return an object based on a view model class. For best results, use AutoMapper to help with this. For example, using the supplier fetched above:

**return** mapper.Map<Supplier, SupplierBase>(supplier);

When returning a collection, return an enumerable collection that’s based on a view model class. Use AutoMapper. For example, using the diplomaPrograms fetched above:

**return** mapper.Map<IEnumerable<Program>, IEnumerable<ProgramBase>>(diplomaPrograms);

**The “deferred execution” idea**

You should be aware that your fetch/query is executed when the results are used. In other words, the statement that uses the data context’s entity set does not necessarily cause a fetch/query at the data store.

Often, the query gets executed only when the fetch/query result is used. In the examples above, when the result is converted (by the AutoMapper statement) into a view-model class based object or collection, that’s when the query gets executed.

This idea is known as “deferred execution”.

This is a subtle point, but it may help explain odd behaviour for programmers who are new to LINQ and data stores, as they practice using LINQ.

**Work on Assignment 3**

Your professor will guide students as we get started on Assignment 3.

Before you leave the room at the end of the time slot, ensure that your professor has checked your work, for the in-class grading part of the assignment.