Section 3 Class Notes

Arrays are reference types. But what they are storing can be either reference or value types.

Formal syntax for array:

T [ ] id;

Int [ ] hours; //uninitialized at this point.

Size is never part of type. Size is determined at runtime.

To create array, use standard New syntax:

Int [ ] hours = new int[ size ]; //size must be greater than or equal to zero.

To access array, use index. Index is zero-based:

Var x = hours[ i ];

Hours[ i ] = 10;

**Collections**

Store 100 student names

String[ ] names = new string [ 100];

All 100 values would be null. It only allocates space for the items.

//get names

Foreach (var name in names){}

System.Collections

-List (default generic – use it)

-Collection (default generic – use it)

-Stack

-Queue

-Dictionary (key – value)

1. Declare an object to store student recs

List<Student>

1. How do you delete the contents of a list?

Clear()

1. What is a generic type?

Generic implementation using arbitrary type.

1. What is an open type?

A type that cannot be created. Generic types, by definition, are open types.

Class List<T>

{

Add(T)

}

An Open Type becomes a Closed Type when you add in the extra type. List<T> is an Open Type. But as soon as you say List<Student>, it is now a Closed Type (and you can create an instance of it).

**Interfaces**

Object Initilization – it’s sole purpose is to initialize an object. It ONLY works with New. Keeps you from having a ton of variables in constructor signature.

Standard object set up:

var product = new Product();

product.Id = Product?.Id ?? 0;

product.Name = \_txtName.Text;

product.Description = \_txtDescription.Text;

product.Price = GetPrice(\_txtPrice);

product.IsDiscontinued = \_chkIsDiscontinued.Checked;

//Object Initializer syntax

//Property must be settable…. And you cannot reference a property on the right side of the assignments.

var product = new Product() {

Id = Product?.Id ?? 0,

Name = \_txtName.Text,

Description = \_txtDescription.Text,

Price = GetPrice(\_txtPrice),

IsDiscontinued = \_chkIsDiscontinued.Checked

};

Along with Object Initializers, you have Collection Initializers.

**Interface** – you don’t need to know how it works. You just use it. Also referred to as contract. The implementation details do not matter to you.

In .NET, it boils down to the members. Given the members, you can call the code. We assume an interface is going to do what is agreed upon. You only have to worry about the signatures.

Foreach is an interface.

All members of an interface are public. Cannot have fields or constructors in an interface. Properties, methods and events are the only things that make sense in an interface.

If you implement an interface, you HAVE to use it. It will not compile if you don’t.

The IEnumerable Validate lists ALL possible errors… so it is a collection.

Quiz questions?

1. Interface

Contract/Abstract. Interface decouple code (look this up)

1. Since an Interface is just a contract, what are valid members of a contract?

Properties

Methods

Events

1. Validation Interface

IValidatableObject (all interfaces start with an I).

They go where a base type would normally go.

**Public class Product : IValidateObject**

Once you specify you are using an interface, you have to use all the ???

Nameof operator – takes any identifier and generates the string literal equivlant. It eliminates????

Yield is only valid in c# a method that returns IEnumerable.? look up

List/Collections are readable and writable.

Arrays are Readable and Replaceable.

IEnumerable<T> is Read only

IEnumerable doesn’t get ALL items (think GetDirectory). It gets items until it finds what you want, then it stops. So it is fast and saves on memory.

Collections are used when you need to modify something.

Use Arrays as a last option.

Implicit Interface Implementation (look this up)

An abstract class is a class that you cannot instantiate. They do have some implementation.

What does implantation mean??????

Quiz Questions?

1. Validate Book if it implements IValidatableObject?

Validator.TryValidateObject(instance, new ValidationContext(instance), Collection<ValidationResult>)

1. Implicit vs explicit interface implementation

Implicit interface implements automagically based upon existing public members.

Explicit interface member

1. What is an abstract class

Base implementation

Must be inherited

DataGridView

Designed to render rows of data.

-Databinding

-WinForms only control – the OS does nothing. The ListView and ListBox are both supported by Windows. Not Winforms. Winforms is a heavy control because of this.

-ASP.Net has the exact same type name. Watch out to make sure you aren’t looking at wrong datagridview when searching on internet,, etc.

-DON’T use Asp.Net GridView or Winforms DataGrid. Those are version 1. Use DataGridView.

-Columns are called DataGridViewColumn – you specify what type of data you are using. Ex. Would be textbox, checkbox, image, etc. Column type decides what user can do with column.

-Rows are called DataGridViewRow. An Array of values. One for each column.

-Cells are DataGridViewCell – raw data stored in cell. Rows are a collection of cells. First row is DataGridViewColumnHeader.

All of these DataGridView…. are types.

-Main type is DataGridView. It has a LOT of events.

-Column Names are fields, and must be unique.

Binding Source – just a wrapper around the data you want to use. That’s it. It is a data type in WinForms. It provides an abstraction layer over the top of the data you want to render. It is what DataGridView wants to work with.