CSHARP 5- CLASSES AND OBJECTS

Kevin.Li@GeorgianCollege.ca

Outline

- OOP
- Class & Object
- Inheritance
- Interface
- Exceptions

STORY

Delivery order

- Freight carrier:
- Owner:
- Consignee:
- Shipping list:
- Delivery instructions:

……

bill of loading

1. Encapsulation, Inheritance, Polymorphism

- **Encapsulation** means that a group of related properties, methods, and other members are **treated as a single unit or object**. -> to reuse codes
- **Inheritance** describes the ability to **create new classes based on an existing class**.
- **Polymorphism** means that you can have **multiple classes** that can be used interchangeably, even though each **class** implements the same properties or **methods** in different ways**.(Poly = many, morph = form) ->polymorphic types and methods**

Find at least one example about the three words

- Encapsulation:
- Inheritance:
- Polymorphism: virtual + override

2 Class and Object

- **A class is** a building **block of OOP**. It is the way to **bind the data** and its logically related **functions** together.
- **A class is** a~~n abstract~~ **data type** that can be treated like any other built in data type.

- Classes describe the **type** of objects… (Generally class itself cannot complete jobs)
- …while objects are usable **instances** of classes.
- Using the blueprint analogy, a class is a blueprint, and an object is a building made from that blueprint. (A house blueprint cannot be lived in…)

2.1 Defining Classes and Objects

- Class is a User Defined Type in .NET
- Object is an instance of a class created with "new" keyword

```
//Declare class A
public class A {
    public void f() {
    }
}
public class B {
    public void g() {
        //Declare object of A
        A a = new A();
        a.f();
    }
}
```

2.2 Methods

- A method is an action that an object can perform. To define a method of a class:

```
class SampleClass
{
 public int sampleMethod(string sampleParam)
  {
     // Insert code here
  }
}
```

- Method can return only a single value or a collection of values

Lab

- Create your class and object to print current time (Instructor shows the steps firstly.)
- Or Test the codes in the last slide (ClassA)

- Would like to make sure everyone succeeded doing this task!

Method overloads

- **A class can have several implementations, or overloads, of the same method that differ in the number of parameters or parameter types**

public int sampleMethod(string sampleParam) {};

public int sampleMethod(int sampleParam) {}

Public int sampleMethod(int param1,int param2){};

- In most cases you declare a method within a class definition. However, both Visual Basic and C# also support extension methods that allow you to add methods to an existing class outside the actual definition of the class.

Real sample in ASP. NET MVC

```
public ActionResult GetAgenies() {
//show all agencies to select and notify
};
public ActionResult  GetAgencies(string jobId) {
//all matched agencies for the current job posting
}
public ActionResult  GetAgencies(string jobId, int topX ) {
//top 10 favorite agencies and matched, and then the remaining favorite
}
// all agencies not in the last list……
```

http://core.direqlink.com/Home/ClientJobRequest/157#

Why overloading, not a new method

- The reason for this is to reduce the risk of the brittle base class problem, where the introduction of a **new method** to a base class could cause problems for consumers of classes derived from it.

(Not required in this class, but think about it.)

2.3 Constructors

- Constructors are class methods that are **executed automatically** when an object of a given type is created. (**Run before any other code**)
- Constructors **usually Initialize the data members**.
- **Support Overloading (Sample: your Win App)**

To define a constructor for a class:

```
public class SampleClass {
        int I;
        public SampleClass() {
        I = 1;
        }
}
```

- If a constructor is not declared , the complier provides one for you.
- The default constructor creates the object but takes not action.
- Member variables are initialized to innocuous values
  - Integers to 0 (Zero)
  - Strings to the empty string, etc.

2.4 Initializers

```
class A
{
private int x = 100; // initializers
}
```

2.5 Destructors

- Destructors are used to destruct instances of classes.
- In the .NET Framework, the garbage collector automatically manages the allocation and release of memory for the managed objects in your application.
- Destructors are still needed to clean up any **unmanaged resources** that your application creates.
- There can be only one destructor for a class.

- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/destructors

```
class Car
{
    ~Car()  // finalizer
    {
        // cleanup statements...
    }
}
```

2.6 The this Keyword

Refers to the current instance of an object

in a Winform app:

```
this.lableName.text = "new text content";
this.pictureBox1.visible=true;
```

2.7 Nested class

- A class defined within another class is called nested.

```
class Container
{
    class Nested
    { // Add code here. }
}
```

To create an instance of the nested class, use the name of the container class followe
by the dot and then followed by the name of the nested class:

```
Container.Nested nestedInstance = new
Container.Nested();
```

2.8 Access modifiers

- All classes and class members can specify what access level they provide to other classes by using access modifiers.

- **`public`**: Access is not restricted.
- `protected`: Access is limited to **the containing class or types derived from the containing class**.
- `internal`: Access is limited to **the current assembly**.
- `protected internal`: Access is limited to types derived from the containing class **or** the current assembly.
- `private`: Access is limited to the containing type.
- `private protected`: Access is limited to the containing class **or** types derived from the containing class within the current assembly

2.9 Static

- Static class
- Static member

Static class

- a static class cannot be instantiated
- can be used as a convenient container for sets of methods that just operate on input parameters and do not have to get or set any internal instance fields.

For example, in the .NET Framework Class Library, the static System.Math

Static members

- Instance members vs Static members

Defining:

```
static class SampleClass {
public static string SampleString = "Sample String";
 }
```

How to access

classInstance.memberName

Vs

ClassName.StaticMemeberName

Use static fields

- For example to keep track of the number of instances

Cat.HowManyCats();// a static method printing

- **Static** methods can access only **static members**.
- A **static class** can only contain **static members**.
- Try to avoid using static fields

2.10 Passing parameters

Passing value types or reference types

Passing value types as reference

Passing value types as reference with the key word: out

Explain and do labs in VS

2.11 Default parameters

```
public void ShowName(int id, bool showAlert=false)
{
        if (showAccount)
                {//Show an extra textbox  }
} //call: ShowName(200) and ShowName(200,false) work


//default values must come at last
/*public void ShowName(bool showAccount=true, int id) cannot be  complied*/
```

Example:

Show specific alert for users that changed the profile.

Default parameters provide opportunities to change your existing codes with extra parameters

2.12 Encapsulating with Properties

```
public int FooProperty { get; set;
}
//so incredibly different than
public int FooField;
```

- Properties combine aspects of both fields and methods.
- To the user of an object, a property **appears to be a field**, accessing the property requires the same syntax.
- To the implementer of a class, a property is **one or two code blocks, representing a get accessor and/or a set accessor.**

The code block for the **get** accessor **is executed when the property is read;**

The code block for the **set** accessor **is executed when the property is assigned a new value.**

A property without a set accessor is considered **read-only.**

A property without a get accessor is considered **write-only.**

A property that has both accessors is read-write.

Why use properties?

Advantages:

- It allows for versioning if later you need extra logic. Adding logic to the getter or setter won't break existing code.
- It allows data binding to work properly (most data binding frameworks don't work with fields).

Lab

- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties#example
- Test the first example
- Copy and debug to make work
- Set a breakpoint at

counter = ++counter + NumberOfEmployees;

Run it and watch the values

3. Inheritance

- Inheritance enables you to create new classes that reuse, extend, and modify the behavior that is defined in other classes.
- The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class.
- **A derived class can have only one direct base class**

Derived

Base

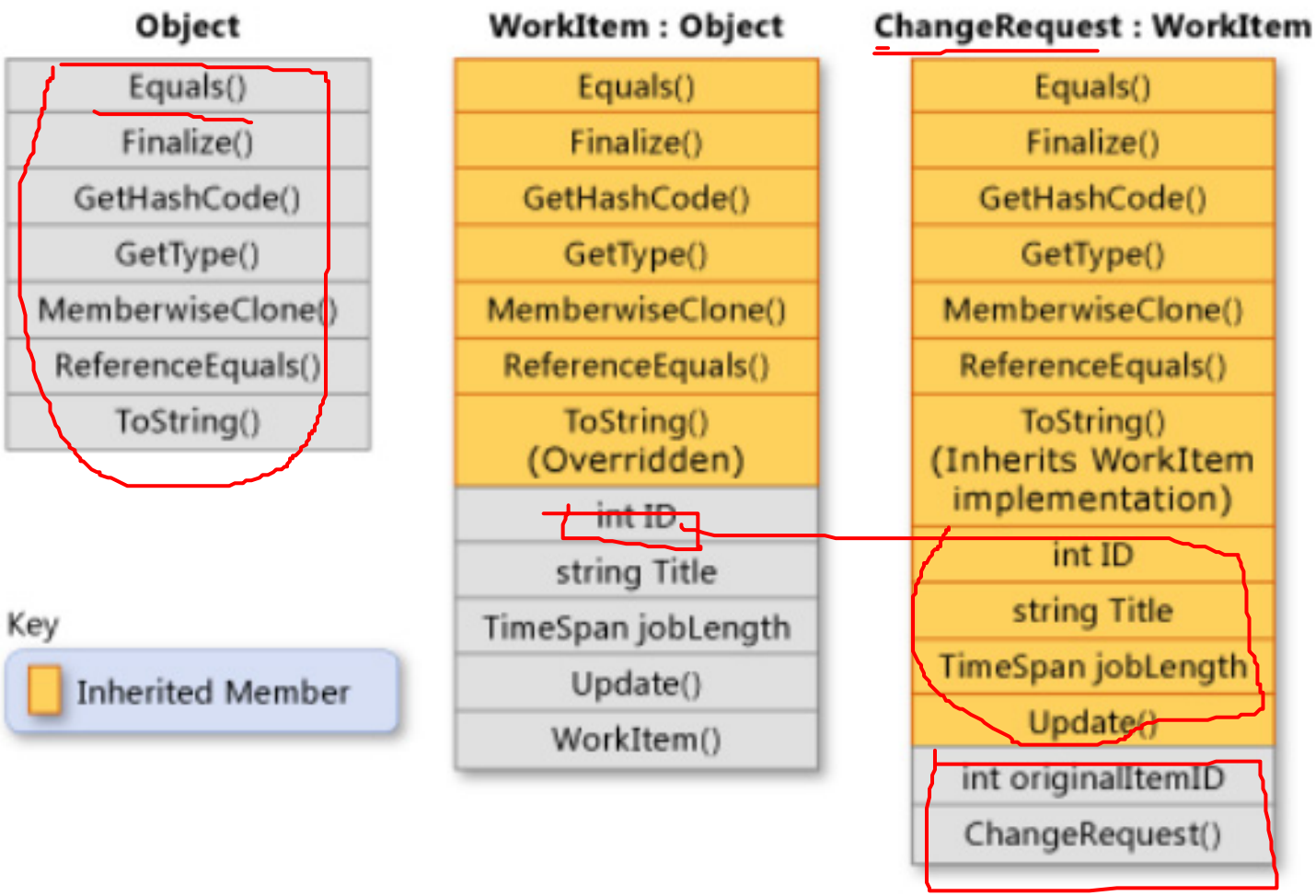
```
class DerivedException : System.ApplicationException
{
    public override string Message
    {
        get { return "An error occurred."; }
    }
}
//
//
//
try
{
    throw new DerivedException();
}
catch (DerivedException ex)
{
    Console.WriteLine("Source: {0}, Error: {1}",
        ex.Source, ex.Message);
}
```

Inherited Member

Base Class

Derived Class

Derived Class



Sample

4.Interfaces

- Interfaces provide a way to achieve runtime **polymorphism.**
- Using interfaces we can invoke functions from different classes through the same Interface reference
- An interface is a reference type that consists of only abstract members.
- **When a class implements an interface, it must provide an implementation for all the members of the interface.**
- **Classes can implement multiple interfaces.**

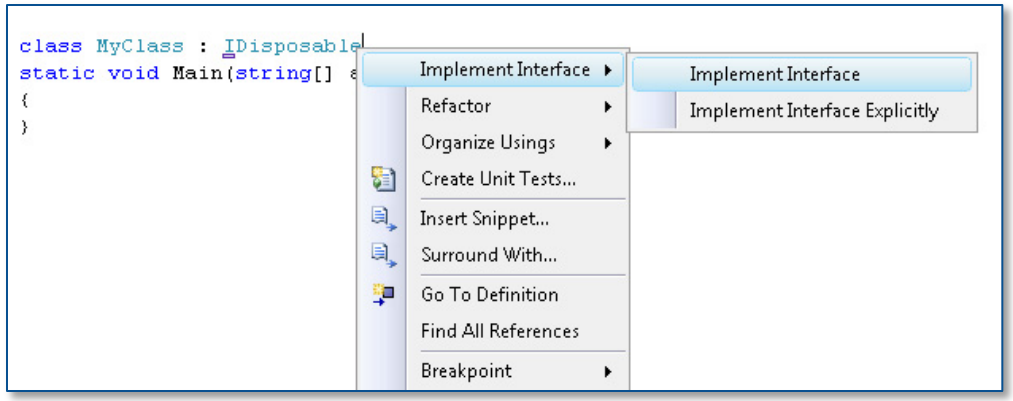
```
class MyClass : IDisposable
    {
        private FileStream _file;

        public MyClass( FileStream fs)
        {
            _file = fs;
        }
        #region IDisposable Members

        public void Dispose()
        {
            _file.Dispose();
        }

        #endregion
    }
```

IDisposable is an interface that provides a single method, Dispose



Commonly Used Interfaces

Type	Description
IComparable	Provides type-specific comparison method, which enables sorting. Types, whose members can be ordered or sorted implement this interface. For instance all numeric and string classes
IDisposable	Provides for disposal of unmanaged objects such as open file. Large objects which consume a lot of resources can implement Idisposable to get rid of the unused resourses
IConvertible	Enables conversion to a base type
ICloneable	Enables creation of a copy of the current class instance
IEquatable	Allows to compare instances of a class for equality.
IFormattable	Provides a ToString() method which enables to convert a value into a formatted string

Creating Interface

```
interface IMessage
{    bool Send();
     string Message { get; set; }
     string Address { get;  set; }
}
class EmailMessage : IMessage
{    public bool Send()
     {
     throw new Exception("not implemented.");
     }
     public string Message
     {    get{ throw new Exception(" not implemented.");    }
          set{ throw new Exception(" not implemented.");    }
     }
     public string Address
     {    get{ throw new Exception("not implemented.");     }
          set{ throw new Exception("is not implemented."); }
     }
}
```

Extracting Interface

- To extract an interface from an already defined custom class :

1) Right-click the class in Visual Studio.
2) Click Refactor and then click Extract Interface.
3) Specify the interface name, select the public members that should form the interface, and then click OK.

Lab & Assignment
=> tnemngissA

Requirement:

1. Create a class that can reverse an input string.

The class contains at least one constructor method to take the input string as the parameter, and another method to return the reversed string, and a private string field to save the input string.

Using the StringBuilder class gets certain scores, not using it will not get them.

2. Create another class or another project to call/use your first class.

Using it in another project will get higher scores.

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

Show your app (Console or Winform or Webform etc..) in the next class and I will mark for you.

String manipulations are typical interview questions. Please find your best solution.(This assignment only asks for the correct result. The interviewers may exam the efficiency.)