

3 Pillars of OOP

CGT 456

Advanced Web Programming, Development, & Database Integration
Lecture 11

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

1

Object Oriented Programming

□ *What is it?*

- Object-Oriented is a paradigm for creating software systems using objects. Objects are tangible and conceptual things we find in the real world. Using OO, the code is broken into modular, reusable chunks called classes. Classes are the "blueprint" for creating instances of objects. These classes can be used throughout an application again and again.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

2

Object Oriented Programming

- OO emphasizes creating reusable, robust software in a way that is easy to understand. By relating programming to the real world, it becomes much easier to use. Some of the characteristics of OO are:
 - **Reusable** - faster, modular development
 - **Robust** - increased quality
 - **Simple** - easy maintenance
 - **Flexible** - easy to modify

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

3

Object Oriented Programming

□ *Concepts Overview*

- There are a lot of concepts involved with OO. But let's just cover the basics for now and maybe this will help you get a better understanding of OO:

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

4

Object Oriented Programming

□ *Concepts Overview*

- *Classes* - A generic blueprint used to create similar objects.
- *Encapsulation* - Hides object data and implementation details.
- *Inheritance* - Allows code reuse by building on existing classes.
- *Polymorphism* - Allows objects to assume many forms.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

5

ENCAPSULATION

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

6

Encapsulation

- Hiding data, but defining properties and methods to let the caller have access to it. Protects the state of the object from other programs and other programs are protected from changes in implementation.
 - *get* and *set* methods that access the property, but you cannot directly access the property itself.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

7

Encapsulation

- Let's say that we simulate a computer. A computer may have a method named on() and another method named off(). When you create an instance of a computer and call its on() method, you are not worried about what happens to accomplish this, you just want to make sure that the state of the computer is changed to 'running' afterwards. Instead of manually turning on the power supply, the fans, the processors, starting the OS, etc, you simply call the on() method, which takes care of all of these tasks for you.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

8

Encapsulation

- *Another example:*
 - A contact list
 - Contact
 - name
 - address
 - phoneNumber
 - getName()
 - setName(string cName)
 - getAddress()
 - setAddress(string address)
 - getPhoneNumber()
 - setPhoneNumber(string phoneNum)

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

9

Encapsulation

```
public class Contact {
    private String phoneNumber;

    public Contact(){
        phoneNumber = "703-567-8860";
    }

    public void setPhoneNumber(String phoneNum){
        phoneNumber = phoneNum;
    }

    public String getPhoneNumber(){
        return phoneNumber.ToString();
    }
}
```

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

10

Encapsulation

- Another example:

```
public String companyName
{
    get { return (String)ViewState["companyName"]; }
    set { ViewState["companyName"] = value; }
}
```

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

11

POLYMORPHISM

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

12

Polymorphism

- Polymorphism by definition means taking many forms.
- In C# it means the ability for classes to share the same methods (actions) but implement them differently.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

13

Polymorphism

- This is the concept where inherited objects "know" what methods they should use, depending on their position in the inheritance chain.
 - House
 - BuildHome()
 - TempestHomes
 - BuildHome()
 - BeazerHomes
 - BuildHome()
 - PrairieViewHomes
 - BuildHome()

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

14

Polymorphism

- A "House" object will know that if it is a BeazerHomes object, it must call the BuildHome method in the BeazerHomes class rather than the one in the House class. You do not need to know what class an object actually belongs to in the inheritance chain when you send it a request.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

15

Polymorphism

- *Another example:*
 - Shape
 - draw()
 - Rectangle
 - draw ()
 - Circle
 - draw ()
 - Polygon
 - draw ()

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

16

Polymorphism

- Rectangle.draw() has a different *implementation* than Shape.draw() or Circle.draw(). The same can be said for each of the other objects.
- Why not have one called square?
 - *Because a square is just a specific form of a rectangle.*

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

17

INHERITANCE

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

18

Inheritance

- The concept of deriving classes from a base class where the derived class "is a" one of the base class (e.g. a Manager "is a" Employee). Inherited classes are always more specialized than their base (parent) classes, have at least as many members (although the behavior of individual members may be different) and often have new methods that have no counterpart in the base class.

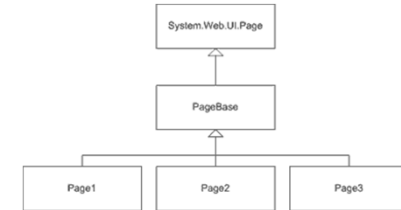
2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

19

Inheritance

- All ASP.NET pages should **derive** (be created) from System.Web.UI.Page



2/14/2012

20

Inheritance



- Page2 "is a" BasePage "is a" Page.
- Page3 "is a" BasePage "is a" Page.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

21

AGGREGATION

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

22

Aggregation

- A subset of Inheritance is **Aggregation**.
- Aggregation is sometimes referred to as the fourth "Pillar of OOP," but it should never be confused as one of the 3 Pillars.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

23

Aggregation

- **Aggregation**, where the method of an inner component is to be directly exposed to the outside world. An object takes the inner object's interface, and presents it as its own. In Aggregation, a host object acts as a liaison between the outside world and an inner object.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

24

Aggregation

An example:

Object 1

```
Public Function GetValue(intID as Integer) as String
    GetValue = Object2.GetValue(intID)
    'we can even add more functionality here
End Sub
```

Object 2

```
Public Function GetValue(intID as Integer) as String
    'get value from database
    'set GetValue = the returnable string
End Sub
```

Object 1 aggregates the functionality of Object 2 by creating a private copy and calls its method internally, presenting it as its own.

2/14/2012

CGT 456
Copyright © 2002-Present Ronald J. Glotzbach

25