3 Pillars of OOP

CGT 456

Advanced Web Programming, Development, & Database Integration

Lecture 11

Object Oriented Programming

- *What is it?*
  - Object-Orientation 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.

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

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:

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.

ENCAPSULATION

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.

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.

Encapsulation

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

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();
    }
  }
```

Encapsulation

- Another example:

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

POLYMORPHISM

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.

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()

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.

Polymorphism

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

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.*

INHERITANCE

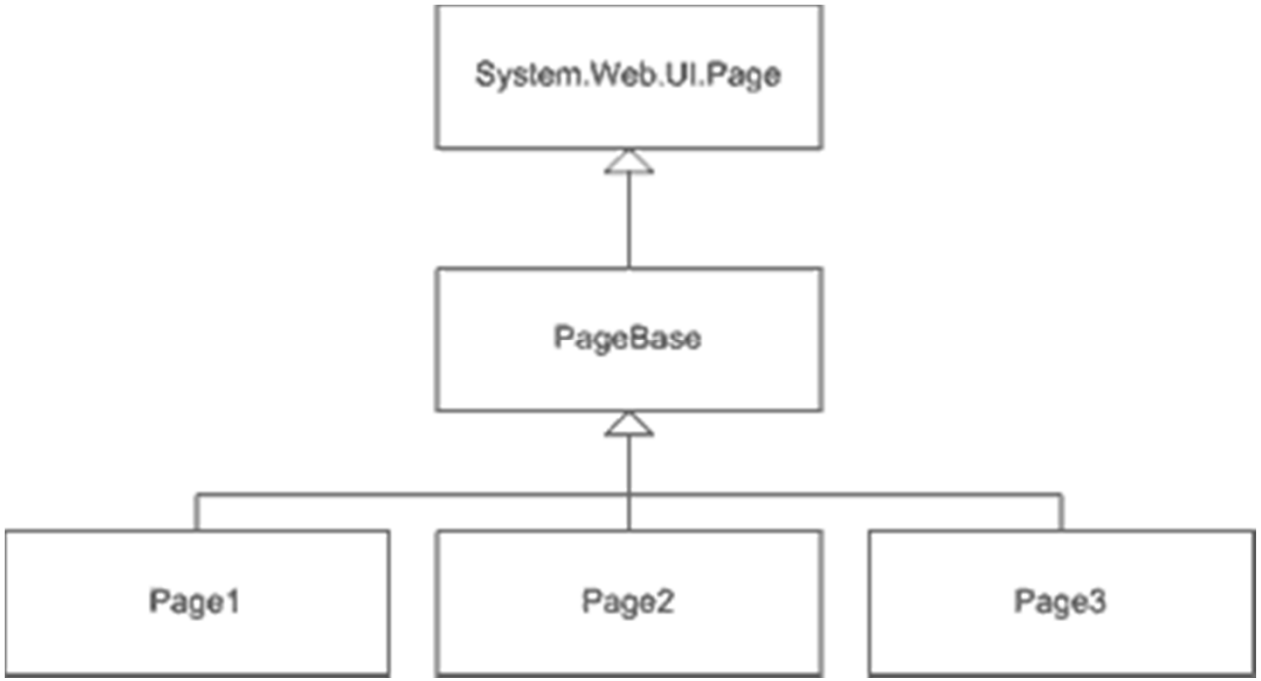
Inheritance

- 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.

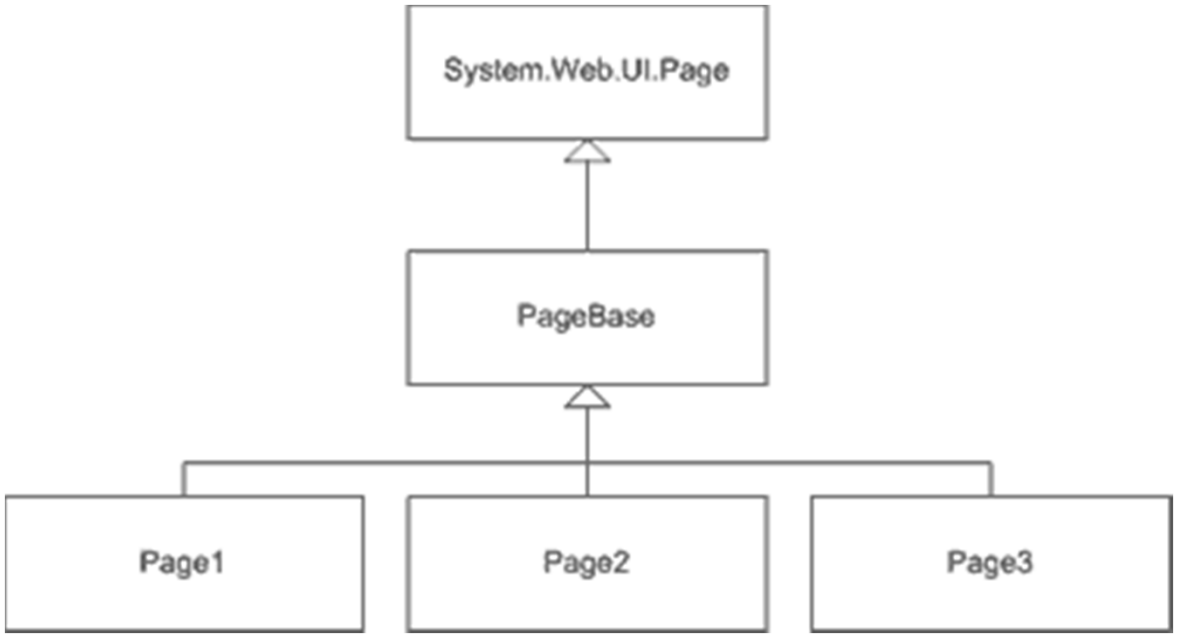
Inheritance

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

  public class BasePage : System.Web.UI.Page



Inheritance



- Page2 “is a” BasePage “is a” Page.
- Page3 “is a” BasePage “is a” Page.

AGGREGATION

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.

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.

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.