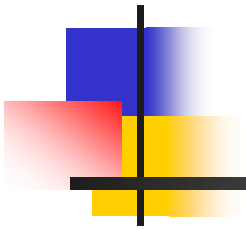
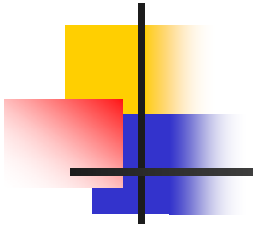


Using ADO.NET II

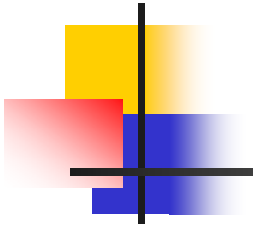
Textbook Chapter 14



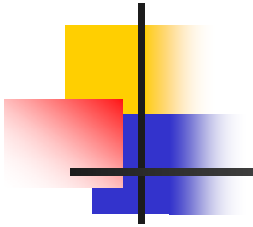
Getting Started

- Last class we started a simple example of using ADO.NET operations to access the Addresses database table:
  - Permit user to look up addresses by last name.
  - TextBox for user input.
  - Do query for address having that last name.
  - Display results or "Not Found" message.

Getting Started



- Download a slightly improved version of the example as of the end of last class:
- http://www.cse.usf.edu/~turnerr/Web_Application_Design/Downloads/2012_06_12_In_Class/
  - I have added the rest of the output controls and put all output controls into a table so that they are aligned.
- Open website in Visual Studio
  - Drill down to the real website folder!
  - Or extract it from the enclosing folder(s)
- Build and run.



Connection String

- Function Setup_Connection will require a *connection string*.
- Rather than hard coding the connection string in your C# code, it is good practice to put it into web.config.
  - web.config can be edited without needing to modify the app code.
- .NET provides a convenient way to retrieve connection strings from the config file.
  - The WebConfigurationManager class

Connection String in web.config

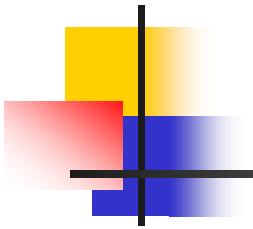
```
<connectionStrings>
    <add name="AddressTable"
         connectionString="server=scorpius.eng.usf.edu;
                           Initial Catalog=wpusr40;
                           User=wpusr40;
                           Password=xxxxx"/>
</connectionStrings>
```

Setup_Connection()

Name in web.config

```
    private static SqlConnection Setup_Connection()
    {
        String connection_string =
WebConfigurationManager.ConnectionStrings["AddressTable"].ConnectionString;

        SqlConnection cn = new SqlConnection(connection_string);

        cn.Open();
        return cn;
    }
```

WebConfigurationManger requires "using System.Web.Configuration;"



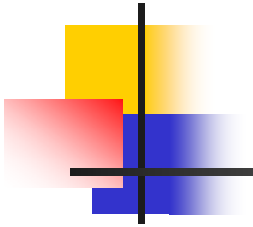
Get_Reader()

```
private static SqlDataReader Get_Reader(string last_name,
                                        SqlConnection cn)
{
    SqlCommand cmd = new SqlCommand();
    cmd.CommandText = "SELECT * FROM Addresses " +
                      "WHERE Last_Name='" + last_name + "'";
    cmd.Connection = cn;
    return cmd.ExecuteReader();
}
```

CAUTION

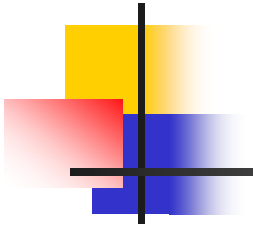
Splicing a command string together like this is NOT good practice.

A little later we will see why and what to do instead.



Using the Query Result

- The SqlDataReader object is similar to a C# StreamReader (or C++ ifstream):
  - An object that we can use to get the query results
- **`Read()`** method makes next line available.
  - Returns true if successful.
- We can then access items in the current line using the column name as an indexer.
  - Example: **`rdr["Last_Name"]`**



Class Address

- Class Address is responsible for knowledge of the structure of the database table.
  - Column names only.
    - No knowledge of how to do a query.
  - Let it extract the individual items from the query result.

- Pass the SqlDataReader object to the constructor.
- Constructor initializes Address object with query results.

Class Address Constructor

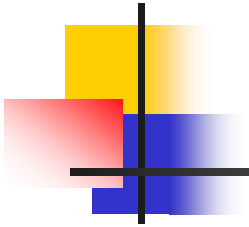
```
using System.Data.SqlClient;
...

public Address(SqlDataReader rdr)
{
    id =          (int)    rdr["ID"];
    last_name =   (string) rdr["Last_Name"];
    first_name =  (string) rdr["First_Name"];
    address1 =    (string) rdr["Address1"];
    address2 =    (string) rdr["Address2"];
    city =        (string) rdr["City"];
    state =       (string) rdr["State"];
    zip_code =    (string) rdr["Zip_Code"];
}
```

Use column name as indexer value for the SqlDataReader.

Typecast the result as the appropriate C# type.

Process the Query Results

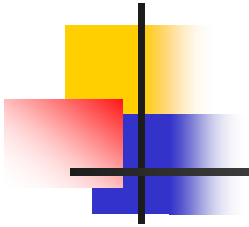
```
public static Address Get_Address(    string last_name,
                                  out string error_msg)
{
    SqlDataReader rdr = null;
    SqlConnection cn = null;
    Address adr =  null;
    error_msg = "";
    try
    {
        cn = Setup_Connection();
        rdr = Get_Reader(last_name, cn);
        if (rdr.Read())
        {
            adr = new Address(rdr);
        }
        else
        {
            error_msg = "Lookup failed";
        }
    }
    ...

    return adr;
}
```



Real Event Handler

Replace the stub in Default.aspx.cs.

```
protected void btnLookup_Click(object sender, EventArgs e)
{
    string error_msg;
    Address adr = Query.Get_Address(tbInput.Text,
                                    out error_msg);
    if (adr != null)
    {
        Display_Results(adr);
    }
    lblMessage.Text = error_msg;
}
```

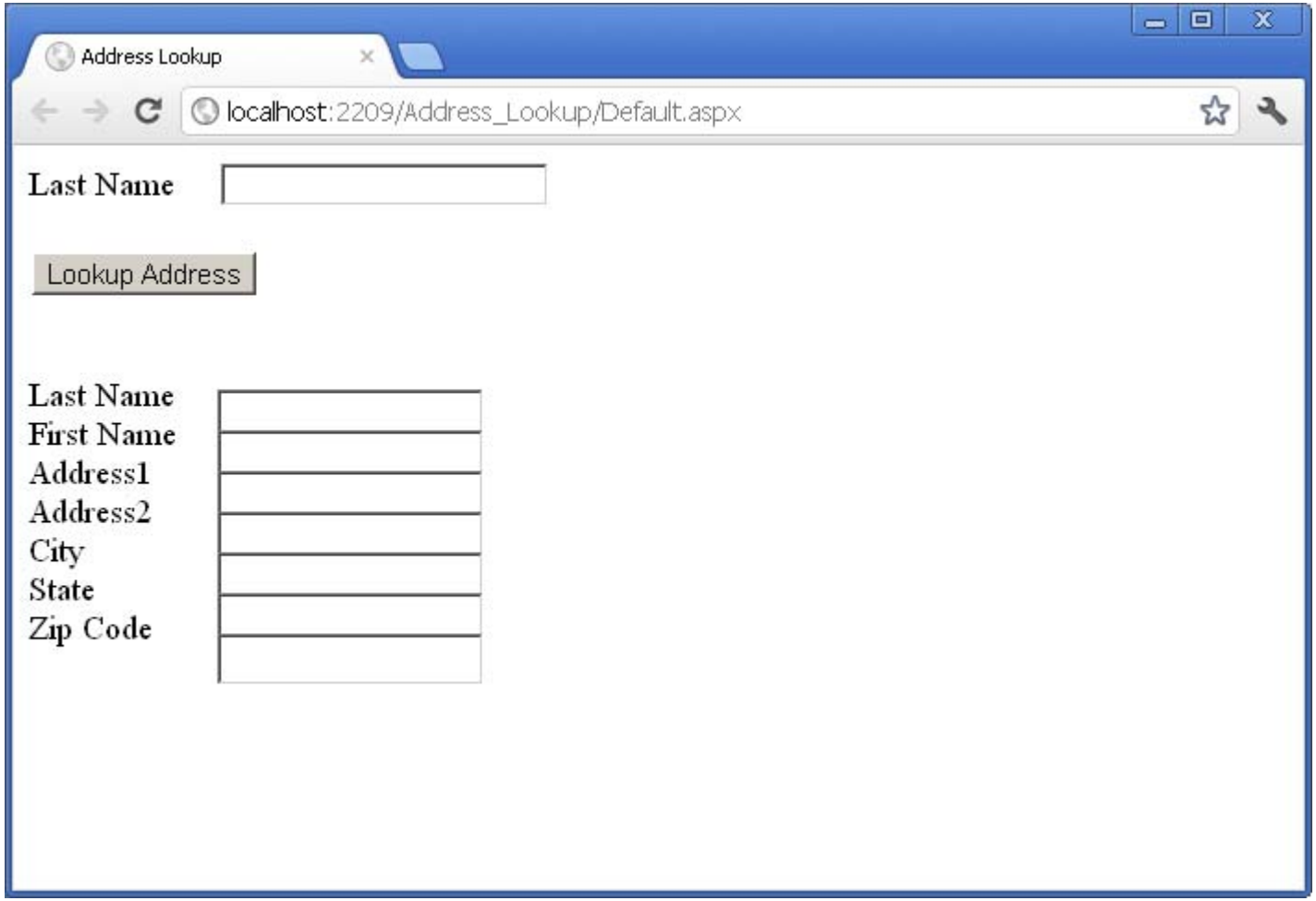


Display_Results()

```
protected void Display_Results(Address adr)
{
    tbLastName.Text = adr.Last_name;
    tbFirstName.Text = adr.First_name;
    tbAddress1.Text = adr.Address1;
    tbAddress2.Text = adr.Address2;
    tbCity.Text = adr.City;
    tbState.Text = adr.State;
    tbZipCode.Text = adr.Zip_code;
}
```

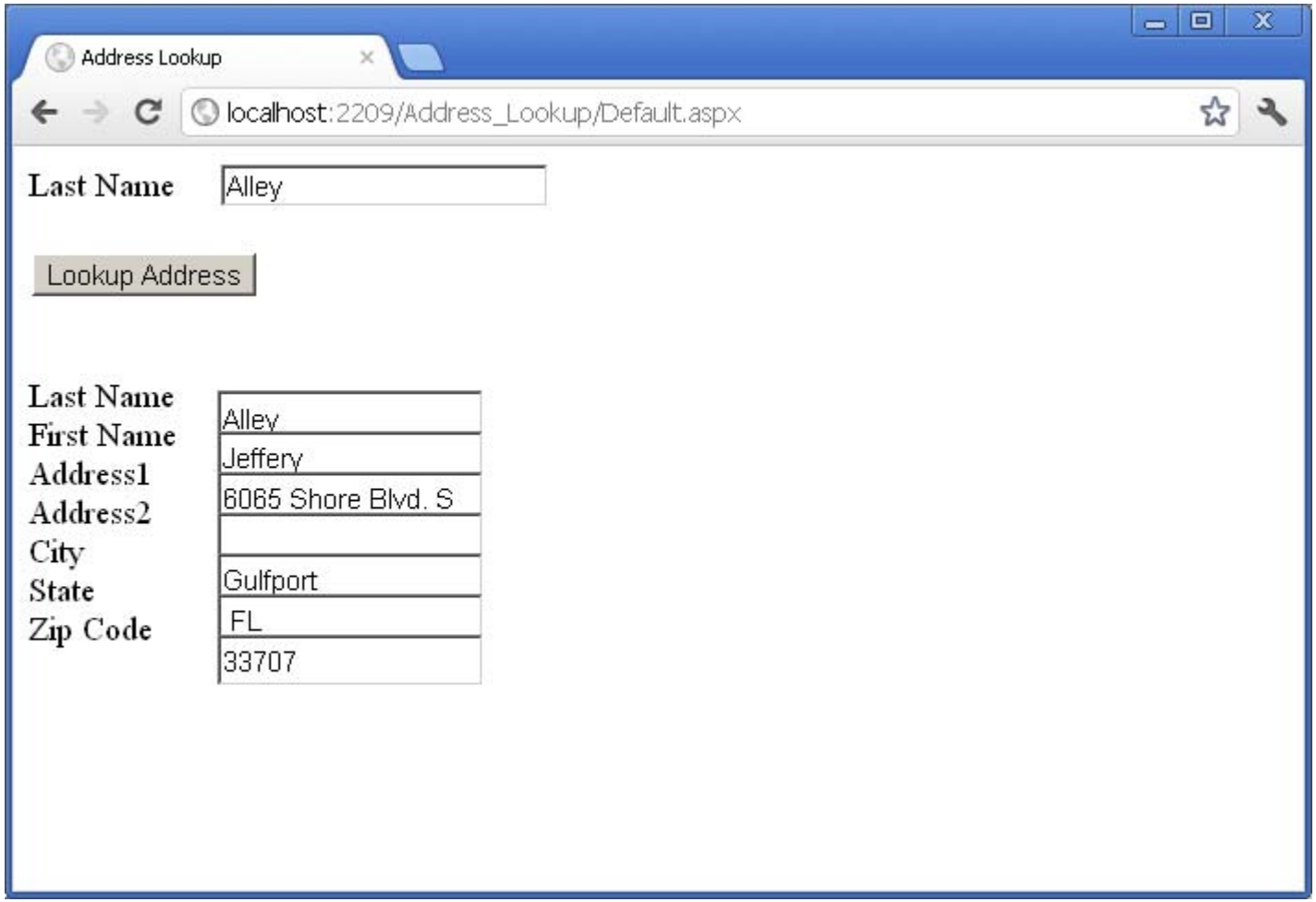
Build and run.

Initial Page



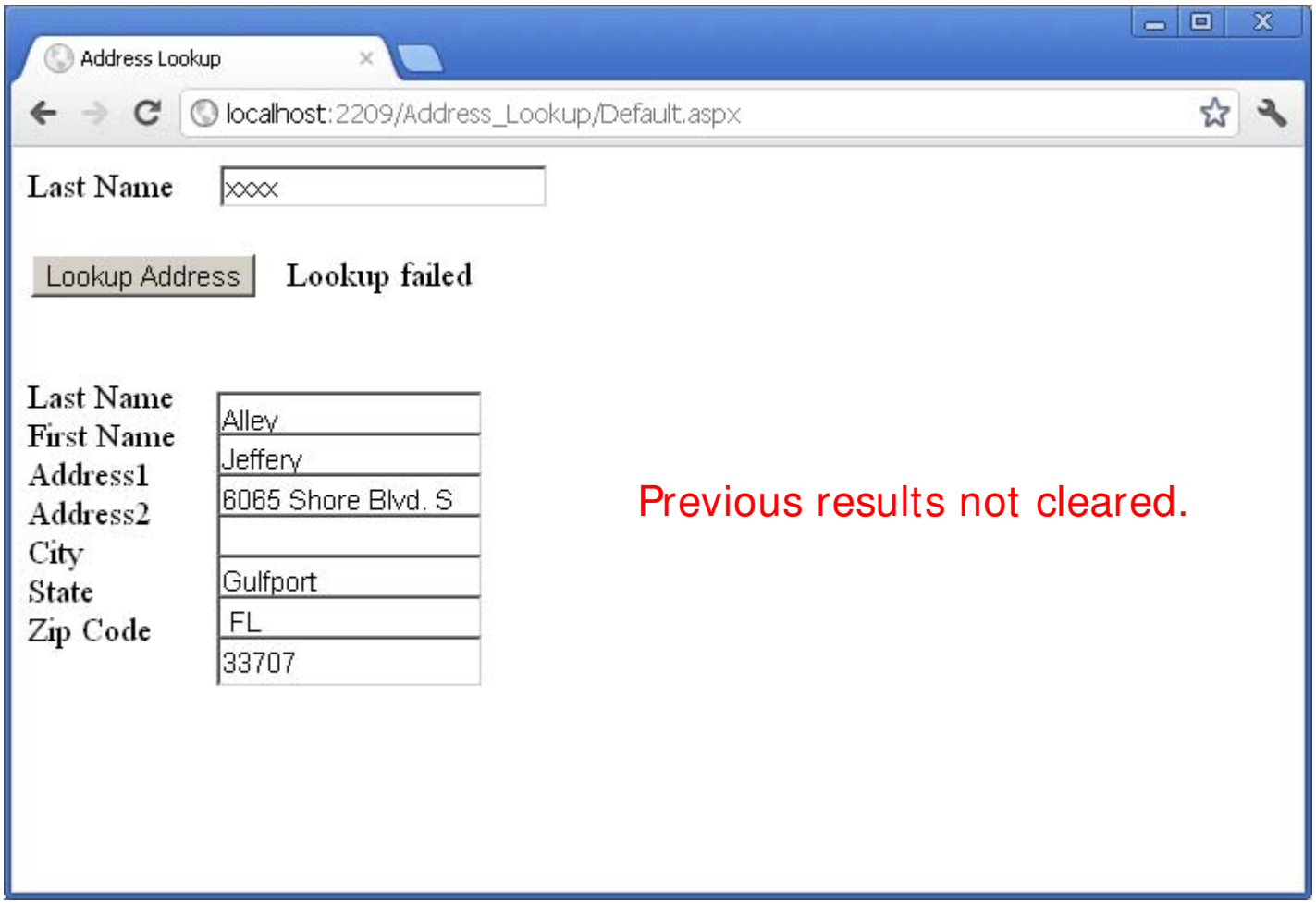
Enter a name and click Lookup Address.

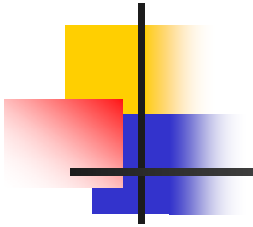
Successful Lookup



Try an unsuccessful lookup.

Unsuccessful Lookup





Clear Previous Results

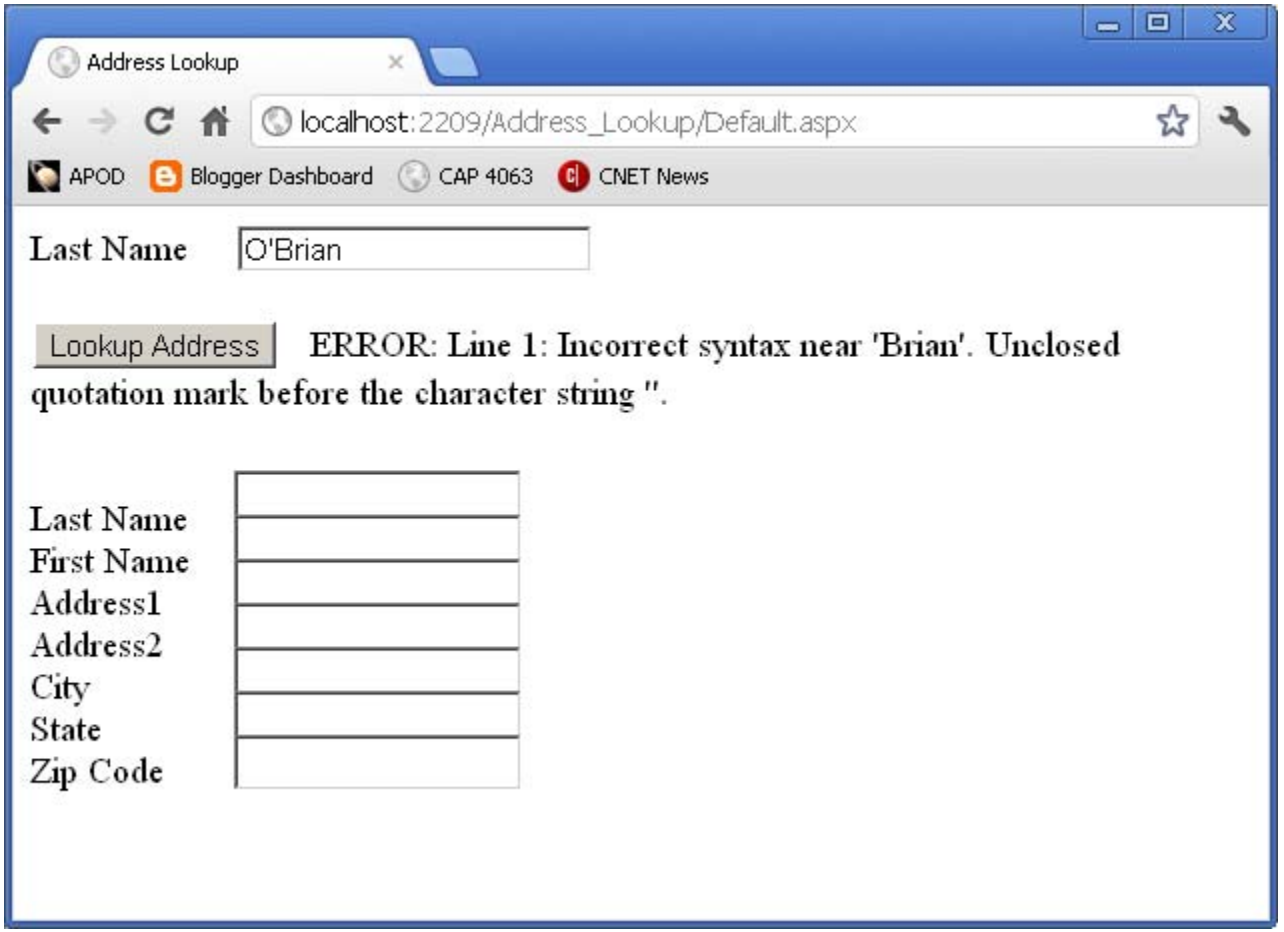
Default.aspx.cs

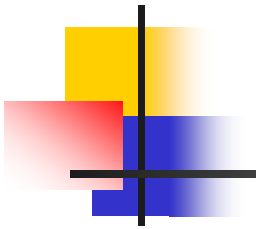
```
protected void Page_Load(object sender, EventArgs e)
{
    lblMessage.Text = "";
    tbLastName.Text = "";
    tbFirstName.Text = "";
    tbAddress1.Text = "";
    tbAddress2.Text = "";
    tbCity.Text = "";
    tbState.Text = "";
    tbZipCode.Text = "";
}
```

Try unsuccessful lookup following a successful lookup again.

A Serious Problem

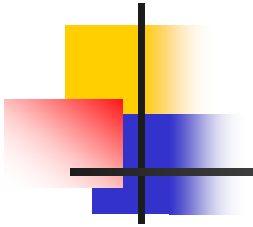
Look up last name O'Brian.





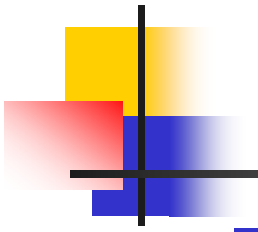
A Serious Problem

- This is the reason you should not build a command string by splicing in user input.
- An apostrophe (single quote) in the user input terminates the last name string, leaving the rest of the input to be interpreted as more command.
- Syntax error in this case.
- Also makes the app vulnerable to a *SQL Injection Attack.*



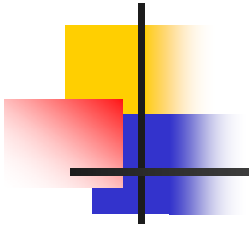
SQL Injection Attack

- A hacker can concatenate his own SQL command after an apostrophe in user input.
  - Potentially execute *any* SQL command.
  - Can take over your database.
  - Destroy your data.
  - Worse, steal it without your knowing.



Defensive Measures

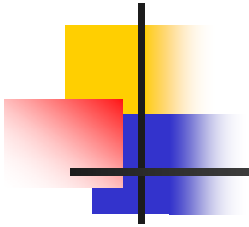
- One defensive measure is to validate the user input.
  - Only accept expected inputs.
- Scan for single quotes in user input and replace them with two single quotes.
  - The SQL Server treats two consecutive single quotes as an escape sequence.
    - Puts one single quote into the command.
    - Does not terminate the string.
- These defensive measures apply to *any* SQL server.



Parmeterized Commands

- ADO.NET provides a better solution:
  - Parameterized Commands

- Rather than splicing together strings, include *parameters* in the command string.
  - Placeholders to be filled in at run time.
  - Set parameter values from user input.
  - Strings as parameter values are not enclosed in single quotes.
    - Will not terminate a string even if they contain single quotes.



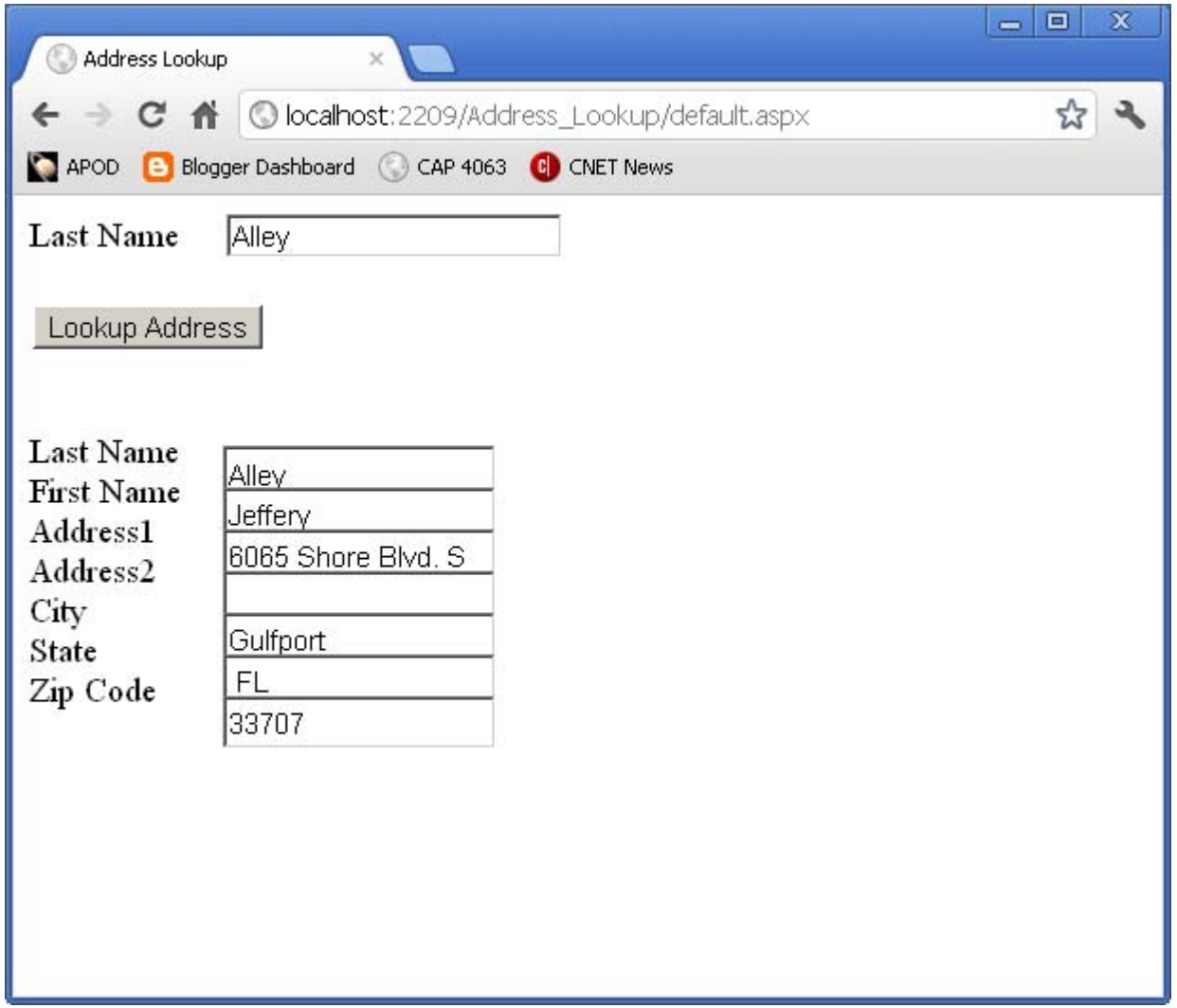
Parmeterized Commands

- The @ symbol in front of a word in a command string in a SqlCommand object's CommandText property identifies the word as a parameter.
  - This only applies to ADO.NET.
  - It is not part of the SQL language.
- Parameter value must be supplied to the SqlCommand object before the command is executed.

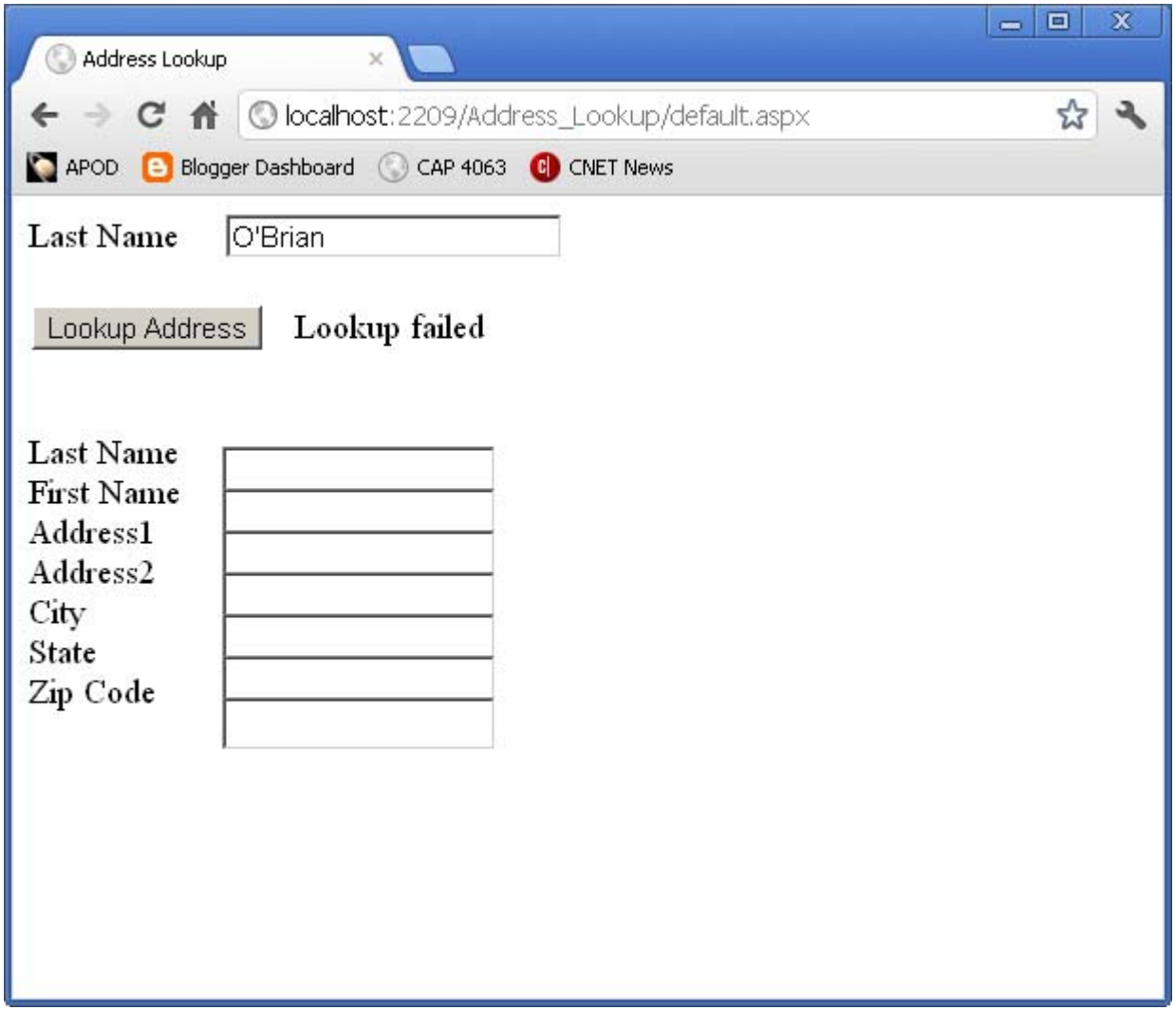
A Parameterized Command

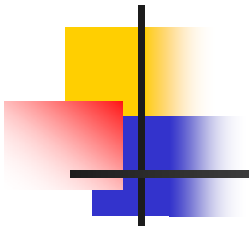
```
private static SqlDataReader Get_Reader(string last_name,
                                        SqlConnection cn)
{
    SqlCommand cmd = new SqlCommand();
    //cmd.CommandText = "SELECT * FROM Addresses " +
    //                  "WHERE Last_Name='" + last_name + "'";

    cmd.CommandText = "SELECT * FROM Addresses " +
                      "WHERE Last_Name=@last_name";

    cmd.Parameters.AddWithValue("@last_name", last_name);

    cmd.Connection = cn;
    return cmd.ExecuteReader();
}
```

Successful Lookup



Input Containing an Apostrophe

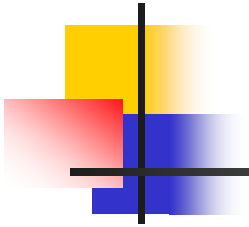




An Ironclad Rule

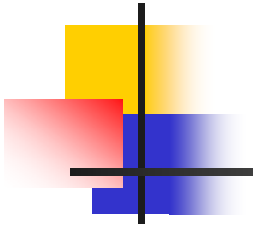
- Never splice user input into a command string.

- Use a command parameter instead.



Loose Ends

- What if there are multiple rows with the same last name?



Summary

- Apply the principles of OOD to web apps.
  - Let the Page class be *just* user interface.
  - Entity class for contents of a database table.
  - Collect query code in a static class.
- Classes from ADO.NET provide easy access to a database table.
  - SqlConnection
  - SqlCommand
  - SqlDataReader
- Use parameterized commands rather than splicing user input into command strings.

End of Presentation