1 **Input Validation**

CST272—ASP.NET

2 **ASP.NET Validation Controls**

- Validation server controls are used to compare input controls (e.g. TextBoxes), or parts of controls, to a "validation rule":
  - Rule may require that the control contain any value, or require a specific form of data such as alphabetical or numeric.
  - The rule may specify that the data must be contained within a range of two values.
  - The rule may be very specific and require formatting such as uppercase letters and periods.

3 **Validation Types (Page 1)**

- Types of validation include :
  - Required Field Input Validation—to ensure that a value has been entered (field not blank)
  - Data type validation—usually for numeric data to ensure that the data *is* numeric
  - Range input validation—to validate that a numeric entry falls within a required range

4 **Validation Types (Page 2)**

- Types of validation include (*con.*):
  - Comparison validation—usually for numeric data to validate that it is equal to, greater than, less than, etc. another value (literal or variable)
  - Pattern validation—for string data to ensure that the characters follow a predetermined pattern, e.g. telephone number or e-mail address

5 **The Validation Controls (Page 1)**

- The ASP.NET validation server controls are:
  - RequiredFieldValidator—Makes sure a form field is *not left blank*
  - CompareValidator—Compares field with other values or values of other controls using *relational operators* (=, >, >=, <, <=, < >)
  - RangeValidator—Makes sure a field's value falls between a given *range* (two numeric literals)

6 **The Validation Controls (Page 2)**

- The ASP.NET validation server controls are (*con.*):
  - RegularExpressionValidator—Evaluates the data against a *string pattern*
  - CustomValidator—Evaluates data against a custom criteria as defined in a *programmer-defined method*

7 **The Validation Controls (Page 3)**

- Validation control *inheritance*:
  - Most validation controls inherit *directly* from the BaseValidator class which inherits from class Label
    - Therefore a custom *error message* displayed using a BaseValidator "is a" Label

- Validation controls that perform *comparisons* inherit directly from BaseCompareValidator base class which inherits from BaseValidator class

8 **The Validation Controls (Page 4)**

- Validation controls perform validation on the *client-side* …
  - They generate JavaScript code in the HTML document which performs the validation processing

9 **The Validation Controls (Page 5)**

- Each validation control only can be used for validating a *single input object*
- However more than one validation control may be applied to the same input object, e.g.
  - To perform both *required field input validation* as well as *comparison validation* on a single input object, e.g. the same TextBox, requires two validation controls

10 **Validation and the "web.config" File (Page 1)**

- Since Visual Studio 2015, it has been necessary to add a key value in the web.config file so that validation controls work correctly
- Without this ValidationSettings key, the browser window will crash when the Button control is clicked

11 **Validation and the "web.config" File (Page 2)**

- Format:

```
<configuration>
    …
   <appSettings>
      <add key="ValidationSettings:UnobtrusiveValidationMode"
            value="None" />
    </appSettings>
</configuration>
```

13 **Common Properties (Page 1)**

- Some of the most important properties for the validation controls are:
  - ControlToValidate—specifies the control on the form that is validated by this validation control
  - CssClass—a validation CSS style class variable can be created which the CssClass property points to the style and handles formatting
    - "CssClass" is the ASP.NET equivalent to the HTML's Class attribute

14 **Common Properties (Page 2)**

- Some of the most important properties (*con.*):
  - Display property—how it saves space for the message:
    - Dynamic—space for the validation message is added to the page dynamically only if validation fails
    - Static—space for the validation message is allocated in the page layout whether

there is an error, or not (the default)
- None—validation message is not displayed (if using a "summary window" instead of individual messages)

15 **Common Properties (Page 3)**

- Some of the most important properties (*con.*):
  - ErrorMessage—the actual text message that is displayed if the value of the control is in error (extends from Text property of a Label)

19 **The RegularExpressionValidator**

- The asp:RegularExpressionValidator server control compares user input to a string pattern for validation
- The control's ValidationExpression property compares values to a "regular expression" …
  - A Regular Expression is a rule (in the form of a string) that describes the value using a language that describes *one or more groups of characters*
- Built-in regular expressions may be selected from the Regular Expression Editor
  - Also used for creating Custom expressions

20 **Some ValidationExpression Property Examples**

- Library of sample codes:

  Internet E-Mail Address

  \w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

  Internet URL

  http://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?

  US Phone Number

  \d{3}-\d{3}-\d{4}

  US Zip Code

  \d{5}(-\d{4})?

24 **The CompareValidator**

- The asp:CompareValiditor server control validates using relational operators (=, >, >=, <, <=, < >)
- Properties unique to the control are:
  - Operator—the relational operator (as a word):
    - Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, DataTypeCheck
  - ValueToCompare—value used by the Operator property for the comparison
  - Type—the data type:
    - String, Integer, Double, Date, Currency

26 **The ControlToCompare Property**

- To use the asp:CompareValiditor control to compare an input object to the value in another input object:
  - Set the ControlToCompare property to the input object to which this object is being

compared
- Set the Operator property as usual
- Set the ValueToCompare property to zero (0) which effectively disables it (ControlToCompare value then takes precedence)

27 **The RangeValidator Control**

- The asp:RangeValiditor control validates that user input is between a lower and upper range of values
- Properties unique to this control are:
  - MaximumValue—the highest value in the range
  - MinimumValue—the lowest value in the range
  - Type—the data type (same as asp:CompareValidator):
    - String, Integer, Double, Date, Currency

29 **The Page.IsValid Property**

- Each time the Form is submitted the Page.IsValid property is updated
  - Indicates if all validation controls on the page are valid
- The following checks if the entire Form is valid:

```
if (Page.IsValid)
{
    LabelMessage.Text = "Result: Valid!";
}
```

31 **The CustomValidator (Page 1)**

- If none of the other validation controls are helpful, the asp:CustomValidator control lets the programmer write the validation code
  - Possibilities are virtually endless
- Allows for validation on either server or client side
  - To specify a client script (e.g. written in VBScript or JScript) within the ".aspx" document, enter its name as the ClientValidationFunction property

32 **The CustomValidator (Page 2)**

- Most properties are exactly the same as for other validation controls, e.g.
  - ControlToValidate, ErrorMessage, etc.
- Double-click on the CustomValidator control while in "Design" view to create new event handler *method* for writing the validation algorithm
  - *Handles* the ServerValidate event

33 **The ServerValidate Event (Page 1)**

- Associated with a CustomValidator object
- Will *fire* whenever a *postback* occurs for a page that contains the CustomValidator control
- Its two arguments are:

  object source

– The CustomValidator object
ServerValidateEventArgs args
– An object with two properties, Value and IsValid

34 **The ServerValidate Event (Page 2)**

- Format:
  protected void *methodName* (object source, ServerValidateEventArgs args)
- Example:
  protected void CustomValidatorDay_ServerValidate (object source,
  ServerValidateEventArgs args)

35 **The Value Property (Page 1)**

- A property of the args parameter associated with the ServerValidateEvent of the CustomValidator control
- The Value property contains the value to be validated (from a TextBox or other input object specified as the ControlToValid) returned as type string
- Like all other strings, the Length property of the Value property is the number of characters that the object contains

36 **The Value Property (Page 2)**

- Format:
  args.Value
- Example:
  String day = args.Value;

37 **The IsValid Property (Page 1)**

- A property of the args parameter associated with the ServerValidateEvent of the CustomValidator control
- The IsValid property is assigned a boolean value to indicate if the validation algorithm has determined the input value to be valid or not:
  – True—the result of the algorithm is valid
  – False—the result of the algorithm is invalid so that the control's ErrorMessage will be displayed

38 **The IsValid Property (Page 2)**

- Format:
  args.IsValid
- Example:

```
if (day == "mon")
    args.IsValid = true;
else
    args.IsValid = false;
```

39 **The ValidateEmptyText Property**

- For CustomValidator control, a boolean property that indicates whether or not empty

text in the TextBox or other input object should be validated

- Values:
  - True—the TextBox or other input object is validated to determine if it is empty (is invalid if it is empty)
  - False—the TextBox or other input object is not validated to determine if it is empty

41 **The CausesValidation Property (Page 1)**

- To disable validation for a particular control like a Button, set the CausesValidation property for that control to False
  - For example a "Cancel" or "Clear" Button might be used to close or clear a Form and *bypass* the validation check
  - Default value is True

42 **The CausesValidation Property (Page 2)**

- Format:

  <asp:*ControlType* ... CausesValidation = "True/False">
- Example:

```
<asp:Button id="ButtonCancel" runat="server"
    Text="Cancel"
    CausesValidation="False"></asp:Button>
```

43 **The ValidationSummary Control (Page 1)**

- The ValidationSummary control summarizes in *one location* the error messages from all validators on a Web page ...
  - In a list somewhere on the page
  - In a separate "alert box" (message box)
- Display of each of the *individual* validation controls usually should be turned off
  - Set their Display properties to the value "None"

44 **The ValidationSummary Control (Page 2)**

- Properties:
  - DisplayMode—error messages displays as a list (List) without bullets, a bulleted list (BulletList), or a single paragraph (SingleParagraph)
  - ShowSummary—shows the entire list within the control itself (default is True)
  - ShowMessageBox—displays errors in a separate alert box (default is False)
  - HeaderText—heading message displayed prior to the error listing