1 **Windows Forms Database**

CST242

3 **The MenuStrip Control**

- The MenuStrip control is used to create Microsoft Windows®-style drop-down *menu bars*
- Menu titles are displayed at top of the Form on the menu bar
- Menu items "drop down" from the menu bar

4 **Creating Menu Commands (Page 1)**

- The MenuStrip is selected from the Menus & Toolbars group in the "Toolbox"
  - It appears at the bottom of the work area IDE in the "Component Tray"
  - Menu bar itself automatically is placed at *top* of Form onto the MenuStrip

6 **Creating Menu Commands (Page 2)**

- Each ToolStripMenuItem (the commands on the menu) is an object with its own properties including Name and Text
- The Text property of each menu title and menu item is the *caption* for the menu element and may be typed directly onto the item

7 **Creating Menu Commands (Page 3)**

- The Name property for each ToolStripMenuItem is assigned in the "Properties" window, just the same as any other control
  - A Name will be assigned automatically if the Text property is typed into the menu directly
  - Consists of the Text property value followed by "ToolStripMenuItem"
  - For example if the Text property is "File" the default Name will be "fileToolStripMenuItem"

8 **Creating Menu Commands (Page 4)**

- Many of the most common MS Windows® menu commands include a shortcut key
  - A character (for example the <Ctrl> key) combined with another character executes the command *directly* (or one of the functions keys, e.g. F1)
  - For example, the keystrokes <Ctrl> + P should print the current document; or <Ctrl> + S saves the document
  - Selected by modifying ShortcutKeys property in the "Properties" window

9 **Assigning Code to Menus (Page 1)**

- Each item in a menu is a Visual C# *object*
- Code is assigned to event handler methods for menu items the same way it is to any other control/object
- To run the code, select menu item from the menu bar at *run-time*

10 **Assigning Code to Menus (Page 2)**

- To get to a menu item's Click event (which is its *default* event), double-click the item from the menu during *design time*

- Alternately to assign a new function to multiple menu items, in the "Properties" window select the event and manually type the name for the new shared function

12 **The ToolStrip Control (Page 1)**

- The ToolStrip control is a container that is used for creating a Microsoft Windows®-style *toolbar* with a collection of button items
- A toolbar is used to execute *frequently used* commands (which also might be found in one of the application's *menus*)
  - Provides *one-click* functionality

13 **The ToolStrip Control (Page 2)**

- Found in the Menus & Toolbars group of the "Toolbox"
  - It appears at the bottom of the work area IDE in the "Component Tray"
  - The buttons automatically are placed onto the ToolStrip usually near the *top* of the Form (and often just below the MenuStrip)

14 **The ToolStrip Control (Page 3)**

- Each item on a ToolStrip is a ToolStripButton which may display an *image*, *text*, or both—images may inserted from:
  - A "Local resource" which will need to be installed independently wherever the application runs
  - The "Project resource file" which is a member of the Visual C# project
- Clicking on any item on ToolStrip object triggers an ItemClicked event (for the *entire* ToolStrip)

15 **Setting ToolStrip Properties**

- Some of the properties for the ToolStrip control:
  - Items: The *collection* of ToolStripButtons and other object types on the toolbar
  - Size: By default Height is 25 pixels, Width is the width of the Form
  - ShowItemToolTips: Set to True or False

16 **Adding Items to the ToolStrip**

- The "easier" way to add items to a ToolStrip is to click on the ToolStrip object itself
  - Click the Add menu item button and select an object "type" from the list
  - The object is added as the next item onto the toolbar
- Or the Items property is used to open the "Items Collection Editor" and to create individual buttons and set their properties

17 **ToolStrip Item Types (Page 1)**

- Item type defines *behavior* of each button:
  - Button
    - Click once to send a Click event which executes code
    - Like clicking a menu item
  - ComboBox
    - The same functionality as a stand-alone ComboBox (e.g. Add method,

SelectedIndex property)
- The selected item text appears in the ComboBox Item on the toolbar after it is selected

18 **ToolStrip Item Types (Page 2)**

- Item type defines *behavior* of each button (*con*.):
  - DropDownButton
    - When clicked, displays a drop-down menu below it
    - DropDownItems property to create menu items in the "Items Collection Editor" or enter them manually directly onto the DropDownButton
    - Triggers a Clicked event when any sub-item in drop-down list is selected
  - Separator
    - Vertical line between two items; used for grouping

19 **Properties of the Items Collection (Page 1)**

- Properties for individual objects on the ToolStrip may be set in the Properties window or from within the "Items Collection Editor":
  - Name: E.g. "toolStripButtonFirst"
  - DisplayStyle: None, Text, Image, ImageAndText
  - Image: A path/file or reference of an image to be displayed on the button
    - Better to add each image as a "Project resource file"

20 **Properties of the Items Collection (Page 2)**

- Properties for objects on the ToolStrip (*con*.):
  - Text: To be displayed on the object instead of an image, or object may have both an image and text displayed together
  - TextImageRelation: Where the two elements appear in relation to each other—Overlay, ImageAboveText, TextAboveImage, ImageBeforeText, TextBeforeImage
  - ToolTipText: When the mouse hovers over the object

22 **The StatusStrip Control (Page 1)**

- The StatusStrip control is a *status bar* that usually appears at the bottom of the Form
- It contains StatusLabel controls which are regions called *panels* that may display both text and images

23 **The StatusStrip Control (Page 2)**

- The purpose of a StatusStrip is to display application status and operating system information, e.g.
  - Is the Caps Lock *on* or *off*?
  - System time and/or date
  - What is *current record number* in a database?

24 **The StatusStrip Control (Page 3)**

- Found in the Menus & Toolbars group of the "Toolbox"
  - It appears at the bottom of the work area IDE in the "Component Tray"

- The "panels" that display the "status" information automatically are placed onto the StatusStrip usually at the *bottom* of the Form

25 **The StatusStrip Control (Page 4)**

- The "easier" way to add to add a StatusLabel is to click on the ToolStrip object itself
  - Click the Add menu item button and select an object "type" from the list
  - The object is added as the next item onto the status bar
- Or the Items property is used to open the "Items Collection Editor" and to create individual panels and set their properties

26 **Properties of the Items Collection (Page 1)**

- Properties for individual StatusStrip objects may be set in the Properties window or from within the "Items Collection Editor":
  - Name: e.g. "toolStripStatusLabelRecordNumber"
  - AutoSize: True (width set to text size-to-fit) or False; if set to True, manual Width property has no effect
  - BorderSides: Is there a border and on which sides? None, All, Top, Bottom, Left, Right

27 **Properties of the Items Collection (Page 2)**

- Properties for Items on the StatusStrip (*con.*):
  - BorderStyle: If the BorderSides property is set to a non-None value, its border style may be Raised, Sunken, one of several others, or None
  - DisplayStyle: None, Text, Image, ImageAndText
  - Spring: True or False; does the panel widen to fill in the remaining space on the status bar (usually set to true for one panel to fill in remaining space)

28 **Properties of the Items Collection (Page 3)**

- Properties for Items on the StatusStrip (*con.*):
  - Text: Displayed in panel but only if DisplayStyle property is set to either Text or ImageAndText
  - TextAlign:
  - Width: May be set to a specific size in *pixels* (only functions if AutoSize property is set to False)

30 **The Main Method (Page 1)**

- All Windows Forms (and Console) applications include a "code-only" (no visual Form elements) class file named "Program.cs" within "Solution Explorer"
- This file, created automatically for any new application, contains a method named Main which is the application's "starting point"
  - The *first method* that will execute in a C# project (same as Java)
  -

31 **The Main Method (Page 2)**

- Example:

```
namespace WindowsForms1
```

```
{
    static class Program
    {
        static void Main()
        {
            ...
        }
    }
}
```

32 **The Run Method (Page 1)**

- The Run method is a member of the Application class that returns a task which is queued to execute in the VisualStudio ThreadPool
  - That is which task to execute next
- In the method Main, this method is used to tell VisualStudio what is the first thing the application should do

33 **The Run Method (Page 2)**

- By default in a Windows Forms application, this task will be to *instantiate* the first Form to be viewed in the application
  - Developer can change this to another Form
- When the application begins running, this Form will open and be ready for interaction with the user

34 **The Run Method (Page 3)**

- Format:

  Application.Run(*task*);
  - The *task* is the next operation that the application will execute

35 **The Run Method (Page 4)**

- Example:

```
namespace WindowsForms1
{
    static class Program
    {
        static void Main()
        {
            ...
            Application.Run( new Publisher() );
        }
    }
}
```

36 **ADO's and OLE DB**

- Computer networks need to be able to share *incompatible* data from differing file types:
  - Data from different software companies such as Microsoft, Oracle, Sybase, Informix, etc.
  - Formatted data (e.g. currency format), graphics, audio, video, etc.
  - Data from a variety of applications, e.g. word processing documents, spreadsheets, etc.
  - Data stored on intranets and the Internet including XML data sources

37 **Universal Data Access**

- The solution to need to share data is called universal data access
- At core to universal access is OLE (object linking and embedding) DB (database) object interface
- Accesses data stored in *different formats* …
  - E.g. treats data stored in spreadsheets and databases as if they were in the same format, *even within the same application*

38 **C# and Databases (Page 1)**

- C# can operate on databases from different software companies:
  - Microsoft, Oracle, Sybase, Informix, etc.
- SQL commands are embedded into C# code which enable communication with the DBMS
- Additionally the Toolbox controls can be configured to provide database connectivity

39 **C# and Databases (Page 2)**

- To use OLE DB objects in a C# application, the "OleDb" namespace must be imported into the source code
- Example:

  using System.Data.OleDb;
  - The keyword using implements "importing" in C#

41 **The DataSet Object (Page 1)**

- A dataset is a programmer-defined *object* that contains one or more tables (or subsets of tables)
- Stores *connection information* including in the "App.config" file:
  - DBMS type and version, e.g. Access '97, 2000, 2002/2003, or MS SQL Server
  - The path and filename of the database
  - Links to the tables, fields and stored procedures (queries) to which the object points

42 **The DataSet Object (Page 2)**

- To create and set properties for a DataSet object in "App.config":
  1. From the Project menu select the command Add New Data Source…
  2. In the "Data Source Configuration Wizard" click the Database icon and then the <Next> button
  3. At the next screen click the DataSet icon and then the <Next> button

43 ☐ **The DataSet Object (Page 3)**

- To create and set properties for a DataSet object in "App.config" (*con.*):
  4. Unless selecting an existing data source, click the <New Connection…> button which opens the "Add Connection" dialog window
  5. Click the <Change…> button to select the provider Microsoft Access Database if necessary
  6. Click the <Browse…> button to select the database
  7. Click the <Test Connection> button to verify that the connection to the database is working (optional)

44 ☐ **The DataSet Object (Page 4)**

- To create and set properties for a DataSet object in "App.config" (*con.*):
  8. Click the <OK> button which closes the "Add Connection" window
  9. Leave "Yes, save the connection" checked ☑ and then click the <Next> button
  10. At next window click checkboxes to select database objects (Tables/Fields/Queries); select "Tables"
  11. Click the <Finish> button to create the dataset which is added to "Solution Explorer"

45 ☐ **Bound Controls**

- Controls that can be linked to an ADO DataSet object at run-time
  - Displays data stored in current record (*only one record* may be active at any one time)
- These controls are considered *data-aware*
- Bound controls may include the TextBox, CheckBox, ComboBox, DataGridView, etc.

46 ☐ **A Comprehensive ADO Application**

- ADO .NET objects that *retrieve*, *insert*, *update*, and *delete* records may be created using program code and embedded SQL
- *Advantage* of using the programmed ADO .NET object over the visual data controls from the Toolbox: Programmer has complete control of any of a multitude of operations
- *Disadvantage*: A lot of programming to learn

47 ☐ **The ADO Objects (Page 1)**

- When coding, five (5) objects must be instantiated:
  1. Create a DataAdaptor object reference (facilitates communication with the data source—the database)
  2. Create a Connection object reference (stores connection information including data source type, as well as the path and filename of database)
  3. Create one or more Command object references (defines SQL SELECT, INSERT, UPDATE and DELETE commands that return and update records)

48 ☐ **The ADO Objects (Page 2)**

- When coding, five (5) objects must be instantiated (*con.*):

4. Create a DataSet object reference (stores a set of tables returned by a SQL SELECT command)
5. Create a DataTable object reference (stores *one* table from the DataSet into RAM and is manipulated *offline*)

49 **The OleDbDataAdaptor Class (Page 1)**

- OleDbDataAdaptor is a class used to instantiate a programmer-defined *identifier* (an object variable)
- It *communicates* between the C# application and a database file
- A member of the System.Data.OleDb namespace
  using System.Data.OleDb;

50 **The OleDbDataAdaptor Class (Page 2)**

- Format:
  OleDbDataAdaptor *objectName* = new OleDbDataAdaptor();
- Example:
  OleDbDataAdaptor dataAdaptorPublisher = new OleDbDataAdaptor();
  - OleDbDataAdaptor is the *type*
- Eventually *links* to OleDbCommand objects (SQL commands are the language used to *communicate* with the database)

51 **The OleDbConnection Class (Page 1)**

- The OldeDBConnection class is used to instantiate a programmer-defined object which defines the connection to a database file
- Stores database *connection information* including the type of DBMS driver, as well as path and filename of database file
- Member of System.Data.OleDb namespace
  using System.Data.OleDb;

52 **The OleDbConnection Class (Page 2)**

- Format:
  OleDbConnection *objectName* = new OleDbConnection();
- Example:
  OleDbConnection connectionPublisher = new OleDbConnection();
- Eventually *links* to OleDbCommand object

53 **Connection Strings on the Internet**

- Connection strings for database applications including Windows Forms database apps can be found at:
  - www.connectionstrings.com

54 **Microsoft Access '97/2000**

- Microsoft Access '97 and 2000 with ".mdb" extension uses the Microsoft Jet OLE DB 4.0
- The connection string for "DataDirectory" functionality in which the path will point to the ".../bin/debug" folder of the project is:

• Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=|DataDirectory|*myDatabase*.mdb;
User Id=admin;Password=;

55 **The ConnectionString Property (Page 1)**

• For the OleDbConnection object, the ConnectionString property builds a String which establishes information for the connection including:
  • The DBMS type and version, e.g. Access '97 or 2000, MS SQL Server, Oracle, etc.
  • The path and filename of database

56 **The ConnectionString Property (Page 2)**

• The information items are in the form of *key*=*value* pairs each separated by a semicolon (;)
• The Provider key specifies DBMS type, e.g.
  "Provider=Microsoft.Jet.OLEDB.4.0"

57 **The ConnectionString Property (Page 3)**

• The Data Source key specifies path (location) to the database and its filename, e.g.
  "Data Source=|DataDirectory|Books.mdb"
• Other keys that are part of the connection string for OleDb 4.0 are User ID and Password

58 **The ConnectionString Property (Page 4)**

• Format:
connectionObject.ConnectionString = "*ProviderString*;*SourceString*"
• Example:
connectionPublisher.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|Books.mdb; User ID=admin; Password=";

59 **The ConnectionString Property (Page 5)**

• Format (easier to use connection string from the "App.config" file):
connectionObject.ConnectionString = " Properties.Settings.Default;*ConnectionString*"
• Example:
connectionPublisher.ConnectionString = "
  Properties.Settings.Default.ConnectionStringBooks ";

60 **The OleDbCommand Class (Page 1)**

• The OleDbCommand class is used to instantate a programmer-defined *identifier* (object) which stores a *SQL command* that *executes* to establish the database connectivity
• Command to returns records, or inserts, updates or deletes rows
• Member of System.Data.OleDb namespace
  using System.Data.OleDb;

61 **The OleDbCommand Class (Page 2)**

• Format:

OleDbCommand *objectName* = new OleDbCommand();

- Example:
  OleDbCommand commandPublisherSelect = new OleDbCommand();
- Eventually *links back* to OleDbConnection object and *forward* to OleDbDataAdapter object

62 **The CommandText Property**

- For the OleDbCommand object, CommandText Property is a String that defines a SQL SELECT, INSERT, UPDATE, or DELETE command
- Format:
  commandObject.CommandText = "*SQLCommandString*"/*StringVariable;*
- Example:
  commandPublisherSelect.CommandText = "SELECT * FROM Publisher"

63 **The Connection Property (Page 1)**

- For the OleDbCommand object, the Connection property stores information about the DBMS to which the command references
- Effectively creates the *link* between the Command object and the Connection object

64 **The Connection Property (Page 2)**

- Format:
  commandObject.Connection = *ConnectionObjectName*
- Example:
  commandPublisherSelect.Connection = connectionPublisher;

65 **The SelectCommand Property (Page 1)**

- For the OleDbDataAdaptor object, stores SQL command information to be executed by the DBMS
- Effectively creates *link* between DataAdaptor object and the Command object

66 **The SelectCommand Property (Page 2)**

- Format:
  dataAdaptorObject.SelectCommand = *commandObject*;
- Example:
  dataAdapterPublisher.SelectCommand = commandPublisherSelect;

67 **The DataSet Class (Page 1)**

- The DataSet class is used to instantiate a programmer-defined *identifier* (object) which stores the *collection of tables* ...
  - May store more than one table returned by the OleDbDataAdaptor object
- The Dataset is stored and manipulated locally "offline" in RAM
  - Physical database is updated *after* processing of the data is completed
- Member of System.Data namespace
  using System.Data;
  - Already imported in every new Windows Forms application

68 **The DataSet Class (Page 2)**

- Format:

  DataSet *dataSetObject* = new DataSet();
- Example:

  DataSet dataSetPublisher = new DataSet();
  - Declared as Private in (Declarations) so object will have *module* scope …
  - But *private* from other modules in the application

69 **The Fill Method (Page 1)**

- Method Fill is a method of an OleDbDataAdapter object which:
  - Opens the connection to the data provider (DBMS)
  - Sends the SQL SELECT command as stored in SelectCommand property
  - After the data provider executes the command, populates the DataSet with data from database

70 **The Fill Method (Page 2)**

- Format:

  dataAdapterObject.Fill(*dataSetObject*);
- Example:

  dataAdapterPublisher.Fill(dataSetPublisher);

71 **The DataTable Class (Page 1)**

- The DataTable class is used to instantiate a programmer-defined *identifier* (object) that *points* to a single table from a DataSet
- Member of System.Data namespace

  using System.Data;
  - Already a Windows Forms application reference

72 **The DataTable Class (Page 2)**

- Format:

  DataTable *dataTableObject* = *dataSetObject*.Tables[*index*];
- Example:

  DataTable dataTablePublisher = dataSetPublisher.Tables[0];
  - Reserved word new is not required because this object is instantiated as the result of the *property value* from the Tables collection

73 **The Tables Collection (Page 1)**

- The Tables collection is a member of DataSet object representing the *one or more* tables assigned to it
- A collection is like an *array of properties* for an individual object
- It is the set of tables that is stored in the DataSet object
  - May include more than one table

74 **The Tables Collection (Page 2)**

- Format:
  dataSetObject.Tables["*TableName*"/*index*]
- Example 1:
  dataTablePublisher = dataSetPublisher.Tables[0];
  - The *index* "0" is used since it is the first (and in our case the only) table from the DataSet

75 **The Tables Collection (Page 3)**

- Format:
  dataSetObject.Tables["*TableName*"/*index*]
- Example 2:
  dataTablePublisher = dataSetPublisher.Tables["Publisher"];
  - Here one table (named "Publisher") from a DataSet collection is assigned to the DataTable reference

76 **The DataRow Class (Page 1)**

- The DataRow class is used to instantiate a programmer-defined *identifier* (object) which *points to one record* (row) from a DataTable
- Member of System.Data namespace
  using System.Data;
  - Already a Windows Forms application reference

77 **The DataRow Class (Page 2)**

- Format:
  DataRow *dataRowObject* = new DataRow();
- Example:
  DataRow dataRowPublisher = new DataRow();

78 **The Rows Collection**

- Rows is a member of the DataTable object representing the *one* row returned from it
  - Like Items collection in a DropDownList or ListBox
- Format:
  DataTableName.Rows[*index*]
- Example:
  DataRow dataRowPublisher = dataTablePublisher.Rows[index];
  - The *index* represents the row's position within the DataTable

80 **Items**

- An Item is one member of a DataRow representing the data from *one* column (field) returned from it
- Format:
  dataRowObject["*ColumnName*"/*index*];
- Examples:
  textBoxPublisherCode.Text = dataRowPublisher["PublisherCode"];

```
textBoxPublisherCode.Text = dataRowPublisher[0];
```

83 **The Count Property**

- The Rows.Count Property for the Rows collection of a DataTable object, returns an integer representing number of rows in the table
- Format:
  dataTableObject.Rows.Count
- Example:

```
toolStripStatusLabel1.Text = dataTablePublisher.Rows.Count;
```

84 **Enums** **(Page 1)**

- An enum type is a set of named *constants* that provide *meaningful* names to discrete values that otherwise are *meaningless*
- Underlying values are type int unless they are specified otherwise
- Unless specifically assigned, values are assigned to each constant starting at zero (0)

85 **Enums** **(Page 2)**

- Format:

```
enum EnumName
{
    CONSTANT_1,
    CONSTANT_2,
    ...
};
```

86 **Enums** **(Page 3)**

- Example:

```
enum EditState
{
    NOT_EDITING,
    ADDING,
    EDITING
};
```

  - In this example (the numeric values are meaningless)
    - NOT_EDITING = 0
    - ADDING = 1
    - EDITING = 2

87 **Enums** **(Page 4)**

- Subsequently to use an enum constant, prefix the constant name with the enum name, like public constants from a class
- Format:
  EnumName.CONSTANT
- Example:

```
if (editState == EditState.NOT_EDITING)
```

88 **Enums (Page 5)**

- Then variables can be declared of the enum type which only can store enum constant values
- Format:

  EnumName enumVariable [= *EnumName.CONSTANT*];
  - *EnumName* is the variable "type"
- Example:

```
EditState editState;
editState = EditState.NOT_EDITING;
```

89 **Enums (Page 6)**

- Using an enum *variable* and an enum *constant* in an if statement
- Format:

```
if (enumVariable == EnumName.CONSTANT)
{ ...
```

- Example:

```
if (editState == EditState.NOT_EDITING)
{ ...
```

90 **The State-Machine (Page 1)**

- The concept that any running program must be in a known state at any given time
- When in a *specific* state, it should be possible to go from that state to only another *legal* state:
  - E.g., if the program is in an edit-state, it only should be possible to move to a cancel-state or a save-state (not to an add-state)
- *Strict enforcement* of the state machine will save the user from himself/herself

91 **The State-Machine (Page 2)**

- Valid states in a database system (with legal destinations):
  - Add-state → Save-state or Cancel-state
  - Edit-state → Save-state or Cancel-state
  - Save-state → Idle-state
  - Cancel-state → Idle-state
  - Delete-state → Idle-state
  - Find-state → Idle-state
  - Move-state → Idle-state
  - Idle-state → Add-state or Edit-state or Delete-state or Find-state or Move-state or Exit-state

92 **Idle-State Environment (Page 1)**

- For two or more records, and at the *first* one

- The Menus:
  - File menu *enabled*
  - Edit menu: New, Edit, and Delete *enabled*; Update and Undo *disabled*
  - Find Published *enabled*
- The Buttons:
  - First and Previous *disabled*; Next and Last *enabled*
- The TextBoxes:
  - Readonly is true (*disabled*)

93 ☐ **Idle-State Environment (Page 2)**

- For two or more records, and at the *last* one
  - The Menus:
    - File menu *enabled*
    - Edit menu: New, Edit, and Delete *enabled*; Update and Undo *disabled*
    - Find Published *enabled*
  - The Buttons:
    - First and Previous *enabled*; Next and Last *disabled*
  - The TextBoxes:
    - Readonly is true (*disabled*)

94 ☐ **Idle-State Environment (Page 3)**

- For two or more records, and *any other* record
  - The Menus:
    - File menu *enabled*
    - Edit menu: New, Edit, and Delete *enabled*; Update and Undo *disabled*
    - Find Published *enabled*
  - The Buttons:
    - First, Previous, Next and Last *enabled*
  - The TextBoxes:
    - Readonly is true (*disabled*)

95 ☐ **Add-State and Edit-State Environments**

- Add-State and Edit-State are *the same*
  - The Menus:
    - File menu *disabled*
    - Edit menu: New, Edit, and Delete *disabled*; Update and Undo *enabled*
    - Find Published *disabled*
  - The Buttons:
    - First, Previous, Next and Last *disabled*
  - The TextBoxes:
    - Readonly is false (*enabled*)

96 ☐ **The Enabled Property**

- The Enabled property is a boolean property for a control set so that when true the control is activated and when false it is not
- Format:
  controlObject.Enabled = true|false;
- Example:
  fileToolStripMenuItem.Enabled = false;

97 **The ReadOnly Property**

- The ReadOnly property is a boolean property for a TextBox or other control set so that when true the user may not type in it
- Format:
  controlObject.ReadOnly = true|false;
- Example:
  textBoxPublisherCode.ReadOnly = true;

99 **The ItemClicked Event (Page 1)**

- The ItemClicked event "fires" for a control that contains an Item collection when a user clicks on an item within a control, e.g.:
  - An Item in a ListView
  - A MenuItem in a MenuStrip
  - A Button or other Item on a ToolStrip

100 **The ItemClicked Event (Page 2)**

- Format for a ToolStrip object:
  private void *toolStripObject*_ItemClicked(object sender, ToolStripItemClickedEventArgs e)
- Example:
  private void toolStripPublisher_ItemClicked(object sender, ToolStripItemClickedEventArgs e)

101 **The ItemClicked Event (Page 3)**

- The ToolStripItemClickedEventArgs variable "e" has a ClickedItem property that gets the item on a ToolStrip that was clicked
- Format:
  e.ClickedItem.*Property*
- Example:
  if (e.ClickedItem.ToolTipText == "First")
  { …

102 **The switch Statement (Page 1)**

- The switch block is a C# structure that can be used to implement *linear nested* functionality (e.g. if … else if … else if …)
  - Exactly the same as in Java
- The value of a *single* variable or expression can be tested for multiple "equal to" values

103 **The switch Statement (Page 2)**

- The keyword break terminates execution of the switch structure when a true code block finishes executing
  - Otherwise program execution will "crash" into subsequent cases
- A final optional default case may be specified and executes if all the previous cases are false

104 **Format of switch Structure**

```
switch (testExpression)
{
   case value:
      statement(s) to be executed when
        this case is true go here;
      break;
   case value:
      statement(s) to be executed when
        this case is true go here;
      break;
  [case ... ]

  [default:
      statement(s) to be executed when
        no case is true go here;]
}
```

105 **Example of switch Structure**

```
switch (e.ClickedItem.ToolTipText)
{
   case "First":
      index = 0;
      break;
   case "Previous":
      index--;
      break;
   case "Next":
      index++;
      break;
   default:
      index = dataTablePublisher.Rows.Count - 1;
      break;
}
```

106 **Equivalent of switch**

```
if (e.ClickedItem.ToolTipText == "First")
{
  index = 0;
}
else if (e.ClickedItem.ToolTipText == "Previous")
{
  index--;
}
else if (e.ClickedItem.ToolTipText == "Next")
{
  index++;
}
else
{
  index = dataTablePublisher.Rows.Count - 1;
}
```

107 **Testing for More than One "true" case in a switch**

- Two or more true cases may tested for as follows:

```
switch (this.editState)
{
   case EditState.ADDING:
   case EditState.EDITING:
     setReadOnly(false);
     break;
   ...
}
```

  - Evaluates true if this.editState is equal to either EditState.ADDING *or* EditState.EDITING

111 **The NewRow Method (Page 1)**

- The NewRow method creates new row object "in the format of" the underlying DataTable
- Not initially added to DataTable, but acts as a *buffer* (work area) until data is stored into it

112 **The NewRow Method (Page 2)**

- Format:

  DataRow *dataRowObject* = *dataTableObject*.NewRow();
- Example:

  DataRow dataRowPublisher = dataTablePublisher.NewRow();

113 **The Add Method**

- The Rows.Add method adds the new DataRow object to the existing DataTable (a *new*

row in the table)
- Should be executed only *after* data is stored into columns (fields)
- Format:
  dataTableOject.Rows.Add(*newDataRowObject*);
- Example:
  dataTablePublisher.Rows.Add(dataRowNewPublisher);

114 **The BeginEdit Method**
- The BeginEdit method enables editing of columns in a DataRow
- Temporarily suspends RowChanging event for the row being edited
  - Row will be *updated once* at the end when the EndEdit method executes
- Format:
  dataTableObject.Rows[*index*].BeginEdit();
- Example:
  dataTablePublisher.Rows[index].BeginEdit();

115 **The EndEdit Method**
- Terminates editing of the DataRow object
- Fires (calls) the RowChanging event to update the entire record
- Format:
  dataTableObject.Rows[*index*].EndEdit();
- Example:
  dataTablePublisher.Rows[index].EndEdit();

116 **The CancelEdit Method**
- Cancels editing of DataRow that was initiated by BeginEdit method
- Format:
  DataTableName.Rows(*index*).CancelEdit()
- Example:
  dataTablePublisher.Rows(mintIndex). CancelEdit()

117 **The GetChanges Method**
- The GetChanges method examines an existing DataSet and returns (creates) a *pointer* to those records in the DataSet which have changed (any inserts, updates and/or deletes)
- Format:
  DataSetName.GetChanges()
- Example:
  DataSet dataSetPublisherUpdate = dataSetPublisher.GetChanges();

118 **The Update Method (Page 1)**
- The Update method for the DataAdaptor examines a DataSet (usually *new* from previously executed GetChanges method) for inserted, updated or deleted records
- Executes the appropriate SQL statement(s) to modify underlying data source (e.g.

database)

119 **The Update Method (Page 2)**

- Format:
  DataAdaptorName.Update(*newDataSetObject*)
- Example:
  dataAdapterPublisher.Update(dataSetPublisherUpdate);

120 **The AcceptChanges Method (Page 1)**

- Method AcceptChanges *Commits* (finalizes) changes to a DataSet
  - Any records in the DataSet that previously had been flagged as having been changed now are marked *unchanged*
- Usually called immediately after the execution of method Update has modified the underlying database table

121 **The AcceptChanges Method (Page 2)**

- Format:
  DataSetName.AcceptChanges();
- Example:
  dataSetPublisher.AcceptChanges();

123 **The Delete Method**

- Removes a DataRow from DataTable in RAM
- The DeleteCommand property of DataAdaptor will finalize the operation in the database table when its Update method is called
- Format:
  DataRowName.Delete();
- Example:
  dataRowPublisher.Delete();

124 **The MessageBoxButtons Class (Page 1)**

- MessageBoxButtons is the third argument of MessageBox.Show
- The YesNo value for this argument displays two buttons, “Yes” and a “No” in the MessageBox dialog window

125 **The MessageBoxButtons Class (Page 2)**

- Format:
  MessageBox.Show("*Text_message*", "*Title_Bar_Caption*", *MessageBoxButtons*);
- Example:

```
MessageBox.Show("Really delete title?", "Confirm Deletion",
                                MessageBoxButtons.YesNo);
```

  - Some other values for MessageBoxButtons are OK (which is the default), OKCancel, AbortRetryIgnore

126 **Other MessageBox.Show Parameters (Page 1)**

- The *fourth* argument for the MessageBox.Show method is MessageBoxIcon which

controls which *icon* is displayed in dialog

- Format:

  MessageBox.Show("*Text_message*", "*Title_Bar_Caption*", *MessageBoxButtons*, *MessageBoxIcon*);

  - Values for MessageBoxIcon include Exclamation, Information, Question, Stop

127 **Other MessageBox.Show Parameters (Page 2)**

- The *fifth* argument for the MessageBox.Show method is MessageBoxDefaultButton
- Controls which button in a MessageBox dialog with more than one button is "pre-selected" as *default*
- Format:

  MessageBox.Show("*Text_message*", "*Title_Bar_Caption*", *MessageBoxButtons*, *MessageBoxIcon*, *MessageBoxDefaultButton*);

  - Available values are Button1 or Button2 or Button3

128 **Other MessageBox.Show Parameters (Page 3)**

- An example with MessageBoxIcon.Question and MessageBoxDefaultButton.Button2:

```
MessageBox.Show("Really delete this title", "Confirm Deletion",
    MessageBoxButtons.YesNo, MessageBoxIcon.Question,
    MessageBoxDefaultButton.Button2);
```

129 **The DialogResult Property**

- The DialogResult property most often uses its values to test which button was clicked in a MessageBox dialog window
- Example:

```
if ( MessageBox.Show("Really delete?", "Confirm Deletion", MessageBoxButtons.YesNo
        ) == DialogResult.Yes )
{ ...
```

  - Some values are Yes, No, OK, and Cancel

131 **Parameters (Page 1)**

- A parameterized SQL statement in Microsoft Access uses question marks (?) to denote placeholders for parameter objects
  - They are "anonymous" in parameterized SQL statements
- The parameter is an object (like a *variable*) that can accept different values based upon the logic of an application

132 **Parameters (Page 2)**

- When the OleDbDataAdaptor object's Update command executes, the operations and values are determined *dynamically* ...
  - Determines if it should run the InsertCommand, the UpdateCommand and/or the DeleteCommand
  - Automatically gets variable values from the changed record(s) and substitutes them into SQL statements
- Example:

```
commandPublisherInsert.CommandText = "INSERT INTO Publisher VALUES (?, ?, ?)";
```

133 **Declaring a Parameter (Page 1)**

- Parameter *objects* to be used in parameterized SQL statements are declared and instantiated from the OleDbParameter class
- There should be one parameter object for each parameter in the SQL statement
- Imported from the System.Data.OleDb namespace:

```
using System.Data.OleDb;
```

134 **Declaring a Parameter (Page 2)**

- Each parameter *substitutes* for a placeholder (?) in a parameterized SQL statement (like a *variable*)
- When the SQL statements are executed, the OleDbDataAdaptor *automatically* substitutes *current value* of the parameter into statement
  - Once configured, requires no additional manual coding by the programmer to get the values

135 **Declaring a Parameter (Page 3)**

- Format:

```
OleDbParameter parameterName = new OleDbParameter("name",
        OleDbType.dataType, size, "columnName");
```

  - *parameterName* is the name of the parameter object (it will be substituted into parameterized SQL command later by the Parameter.Add method)

136 **Declaring a Parameter (Page 4)**

- Format:

```
OleDbParameter parameterName = new OleDbParameter("name",
        OleDbType.dataType, size, "columnName");
```

  - *name* is the name of the parameter object
    - Used to access or modify the Value property of a specific parameter when creating a parameter value "on the fly"

137 **Declaring a Parameter (Page 5)**

- Format:

```
OleDbParameter parameterName = new OleDbParameter("name", OleDbType.type,
        size, "columnName");
```

  - *type* must be *consistent* with the data type in physical database table (e.g. MS Access) although it may be different from the actual C# type

138 **Declaring a Parameter (Page 6)**

- Format:

```
OleDbParameter parameterName = new OleDbParameter("name",
        OleDbType.dataType, size, "columnName");
```

  - *size* is the maximum size of data:
    - For a String it should match that in the database table
    - For *numerics* (including dates) and Boolean it should be zero (0) since it will be

inferred from its OleDbType

139 **Declaring a Parameter (Page 7)**

- Format:

  OleDbParameter *parameterName* = new OleDbParameter("*name*",
  OleDbType.*dataType*, *size*, "*columnName*");

  - *columnName* is the name of the column in the underlying database table

140 **Declaring a Parameter (Page 8)**

- Format:

  OleDbParameter *parameterName* = new OleDbParameter("*name*",
  OleDbType.*dataType*, *size*, "*columnName*");

- Example:

  OleDbParameter parameterUpdateName = new OleDbParameter("Name",
  OleDbType.VarChar, 20, "Name");

141 **The Add Method for Parameters**

- The Parameters.Add method adds a parameter to Parameters collection of the OleDbCommand object
- Must be inserted into the SQL command in the *same order* as listed in the statement
- Format:

  commandObject.Parameters.Add(*parameterName*);

- Example:

  commandPublisherInsert.Parameters.Add(parameterUpdateName);

142 **The InsertCommand Property (Page 1)**

- In addition to the SelectCommand property, DataAdaptor objects additionally may store commands to insert, update and delete rows
- The InsertCommand property stores a SQL INSERT command into a DataAdapter object

143 **The InsertCommand Property (Page 2)**

- Format:

  dataAdaptorName.InsertCommand = *commandObjectName*;

- Example:

  commandPublisherInsert.Connection = connectionPublisher;
  dataAdapterPublisher.InsertCommand = commandPublisherInsert;

144 **The UpdateCommand Property**

- The UpdateCommand property stores a SQL UPDATE command into a DataAdaptor object
- Format:

  dataAdaptorName.UpdateCommand = *commandObjectName*;

- Example:

  commandPublisherUpdate.Connection = connectionPublisher;

dataAdapterPublisher.UpdateCommand = commandPublisherUpdate;

145 **The DeleteCommand Property**

- The DeleteCommand property stores a SQL DELETE command into a DataAdapter object
- Format:

  dataAdaptorName.DeleteCommand = *commandObjectName*;
- Example:

  commandPublisherDelete.Connection = connectionPublisher;

  dataAdapterPublisher.DeleteCommand = commandPublisherDelete;

146 **Review: Six Steps to Implement a Parameterized Command Object (Page 1)**

1. Instantiate the OleDbCommand object
2. Assign Connection object to the Command object's Connection Property
3. Assign a "parameterized" SQL command to the CommandText to property of the Command object

147 **Review: Six Steps to Implement a Parameterized Command Object (Page 2)**

4. Instantiate OleDbParameter objects
5. Add Parameter objects to SQL statement in order
6. Assign the Command object to the DataAdaptor object's appropriate *command property* (SelectCommand, InsertCommand, UpdateCommand and DeleteCommand)

148 **Defining the PrimaryKey Property (Page 1)**

- A PrimaryKey must be declared and instantiated before a call to the Find method can execute to return a Row in a DataTable
  - Method Find searches the PrimaryKey object
- A consists of an *array* of one or more DataColumn objects which is a comma delimited list
  - Allows for concatenated primary keys

149 **Defining the PrimaryKey Property (Page 2)**

- Format to declare a DataColumn array (1 column):

  DataColumn[] *dataColumnArrayName* = { *dataTableName*.Columns[*index*] };
- Format to declare a DataColumn array (more than 1 column):

  DataColumn[] *dataColumnArrayName* = { *dataTableName*.Columns[*firstIndex*],
  dataTableName.Columns[*secondIndex*], ... };

150 **Defining the PrimaryKey Property (Page 3)**

- Example of DataColumn declarations (1 column):

  DataColumn[] dataColumnPublisherCode = { dataTablePublisher.Columns[
  "PublisherCode"] };
- Example of DataColumn declarations (2 columns):

  DataColumn[] dataColumnBookAuthor = { dataTableBookAuthor.Columns[
  "BookCode"], dataTableBookAuthor.Columns["AuthorID"] };

152 **Defining the PrimaryKey Property (Page 4)**

- After DataColumn array object is created, it is assigned to the PrimaryKey property of the DataTable
- Format:
  dataTableName.PrimaryKey = *dataColumnArrayName*;
- Example:

```
DataColumn[] dataColumnPublisherCode = {
        dataTablePublisher.Columns["PublisherCode"] };
dataTablePublisher.PrimaryKey = dataColumnPublisherCode;
```

153 **The Find Method**

- The Find method finds and returns a DataRow from a DataTable with a matching primary key
- Format:
  dataTableObject.Rows.Find(*criteria*)
  - The *criteria* should match a *primary key* value from the table (defined in the PrimaryKey property)
- Example:

```
DataRow dataRowPublisher = dataTablePublisher.Rows.Find(publisherCode);
```

155 **The ShowDialog Method (Page 1)**

- The ShowDialog method displays a Windows Forms object from the "Solution Explorer" as a "modal" dialog box
  - "Modal" disables the main or parent window but keeps it visible
  - Requires user to close modal window before returning to parent or activating another window from it
- C# also has a Show method which is *not modal* so users can go back and forth between the two windows

156 **The ShowDialog Method (Page 2)**

- Format:
  formObject.ShowDialog();
- Example:

```
FindPublisherByCode findPublisherByCode = new FindPublisherByCode();
findPublisherByCode.ShowDialog();
```

  - Object must be instantiated from the Form before ShowDialog is called
  - Once Form is instantiated, its methods can be call even before it displays

157 **The Hide Method**

- The Hide method makes a Form object *no longer visible* and gives control back to the Form that called it ...
  - Visible property also may hide Form but it keeps control of the application
- Allows Form's resources (including its properties) to remain in RAM and available to other modules

- Examples:
  this.Hide();
  Hide();

158 **The Dispose Method**

- The Dispose method closes objects and releases unmanaged resources used by the application
- Since it improves performance and optimizes memory, it is a better option than Close
- Format:
  formName.Dispose();
- Example:
  findPublisherByName.Dispose();

159 **The IndexOf Method**

- The Rows.IndexOf method of a data table returns the "index" number of its data row argument
- Format:
  dataTableObject.Rows.IndexOf(*dataRowObject*)
- Example:
  index = dataTablePublisher.Rows.IndexOf(dataRowFindByCode);

161 **The foreach Loop (Page 1)**

- A foreach loop provides an easy way to *iterate* over arrays or other collections *without an index*
- Format in C#:
  foreach (*type variable* in *collection*)
  { ...
- Format in Java:
  for (*type variable* : *collection*)
  { ...
  - Variable *type* must be the same type as the collection
-

162 **The foreach Loop (Page 2)**

- Example in C#:

```
foreach (DataRow dataRowFind in dataTableFind.Rows)
{
    listBoxFind.Items.Add(dataRowFind);
}
```

163 **The FormClosing Event (Page 1)**

- The Form.FormClosing event occurs as a Form is closing but *before* it actually is closed
- When a Form is closed, all resources within the object are released and the Form is disposed

- The series of events that leads to the Form being closed *can be stopped* and the Form remain open by assigning:
  e.Cancel = true;

164 **The FormClosing Event (Page 2)**

- Example:

```
private void Form_FormClosing(object sender, FormClosingEventArgs e)
{
    if (editState == EditState.ADDING || editState == EditState.EDITING)
    {
        e.Cancel = true;
    }
}
```

165 **The Cancel Property (Page 1)**

- The e.Cancel property of the FormClosingEventArgs parameter (the second parameter) is a property used in a FormClosing event
- If assigned the value true, the Form remains open
- Default value is false, so if no value assigned (or false is assigned), Form closes, all resources are released and the Form disposed

166 **The Cancel Property (Page 2)**

- Format:
  e.Cancel = true|false;
  - Default is false
- Example:

```
if (editState == EditState.ADDING || editState == EditState.EDITING)
{
    e.Cancel = true;
}
```

  - If the condition is false, e.Cancel remains *false*

168 **The DataGridView Control**

- The DataGridView control displays contents of DataTable object
- Data is presented in tabular format (in rows and columns) similar to a MS Access Datasheet view
  - Column names are displayed as headers in the first row

169 **Selecting the Data Source**

- In "Design" view the DataGridView (once it is selected) has a small "arrow" icon in its upper-right corner called a "smart tag"
- To select a table for display in the grid:
  1. Click the smart tag to open the "DataGridView Tasks" dialog window
  2. Select the option "Choose Data Source" and find the DataSet that was configured when creating the database connection string

3. "Drill down" to find the table to be displayed in the DataGridView and select it

170 **Properties for DataGridView**

- Dock—if set to the value Fill, the control fills its container (most probably the Form)
- EditMode—if set to EditProgrammatically, values displayed in the grid may not be updated using keyboard input

171 **The Value Property for Parameter Objects (Page 1)**

- Manually stores new current value for a parameter within parameterized SQL statements
- Format:
  CommandObject.Parameters("*Name*"/*Index*).Value = "*String*"/*Value*;
- Example:
  pcmdDetails.Parameters("PublisherCode").Value = pstrPublisherCode;

172 **The Value Property for Parameter Objects (Page 2)**

- So if current value of String variable "pstrPublisherCode" is "BF" ...
- And the parameterized SQL statement reads:
  commandDetails.CommandText = "SELECT * FROM Publisher WHERE PublisherCode = ?"
- The actual statement that will execute is:
  commandDetails.CommandText = "SELECT * FROM Publisher WHERE PublisherCode = 'BF'"