Methods, Properties, and Events for Common Windows Controls

1. TextBox Control

- **Purpose**: Allows users to input text.
- **Common Properties**:
  - `Text`: The text contained in the `TextBox`.
  - `MaxLength`: Specifies the maximum number of characters the `TextBox` can contain.
  - `ReadOnly`: Determines if the `TextBox` is read-only.
  - `Multiline`: Allows the `TextBox` to display multiple lines of text.
  - `PasswordChar`: Masks the input character (useful for password fields).
- **Common Methods**:
  - `Clear()`: Clears the text from the `TextBox`.
  - `AppendText(string)`: Appends the specified text to the current text in the `TextBox`.
- **Common Events**:
  - `TextChanged`: Triggered whenever the text in the `TextBox` changes.
  - `KeyPress`: Triggered when a key is pressed while the `TextBox` has focus.
  - `Leave`: Triggered when the `TextBox` loses focus.

2. Button Control

- **Purpose**: Used to trigger an action or event when clicked.
- **Common Properties**:
  - `Text`: The label text displayed on the button.
  - `Enabled`: Indicates whether the button can respond to user interaction.
  - `Image`: Displays an image on the button.
  - `FlatStyle`: Determines the appearance of the button (e.g., Flat, Popup, Standard).
- **Common Methods**:
  - `PerformClick()`: Simulates a click event on the button programmatically.
  - `Focus()`: Sets input focus to the button.
- **Common Events**:

        - `Click`: Triggered when the button is clicked.
        - `MouseEnter`: Triggered when the mouse pointer enters the button's area.
        - `MouseLeave`: Triggered when the mouse pointer leaves the button's area.

3. CheckBox Control

- **Purpose**: Allows users to select or deselect an option (can select multiple checkboxes).
- **Common Properties**:
    - `Checked`: Indicates whether the `CheckBox` is checked or not.
    - `Text`: The label text associated with the `CheckBox`.
    - `ThreeState`: Indicates whether the `CheckBox` has two or three states (Checked, Unchecked, Indeterminate).
- **Common Methods**:
    - `CheckState`: Returns the current state of the `CheckBox` (Checked, Unchecked, Indeterminate).
    - `Toggle()`: Toggles the state of the `CheckBox` between Checked and Unchecked.
- **Common Events**:
    - `CheckedChanged`: Triggered when the `Checked` property value changes.
    - `Click`: Triggered when the `CheckBox` is clicked.
    - `CheckStateChanged`: Triggered when the state of the `CheckBox` changes.

4. RadioButton Control

- **Purpose**: Allows users to select one option from a group of mutually exclusive options.
- **Common Properties**:
    - `Checked`: Indicates whether the `RadioButton` is selected.
    - `Text`: The label text associated with the `RadioButton`.
    - `GroupName`: Groups `RadioButton` controls together so only one can be selected at a time.
- **Common Methods**:
    - `PerformClick()`: Simulates a click event on the `RadioButton` programmatically.

    - `Focus()`: Sets input focus to the `RadioButton`.
- **Common Events**:
    - `CheckedChanged`: Triggered when the `Checked` property value changes.
    - `Click`: Triggered when the `RadioButton` is clicked.

5. ComboBox Control

- **Purpose**: Provides a drop-down list from which users can select one item.
- **Common Properties**:
    - `Items`: Collection of items in the `ComboBox`.
    - `SelectedIndex`: The zero-based index of the currently selected item.
    - `SelectedItem`: The currently selected item in the `ComboBox`.
    - `DropDownStyle`: Determines whether the `ComboBox` is editable or read-only (`DropDown`, `Simple`, `DropDownList`).
- **Common Methods**:
    - `AddItem(object)`: Adds an item to the `ComboBox`.
    - `RemoveItem(object)`: Removes an item from the `ComboBox`.
    - `Clear()`: Removes all items from the `ComboBox`.
- **Common Events**:
    - `SelectedIndexChanged`: Triggered when the selected item in the `ComboBox` changes.
    - `TextChanged`: Triggered when the text in the editable `ComboBox` changes.
    - `DropDown`: Triggered when the drop-down portion of the `ComboBox` is shown.

6. GroupBox Control

- **Purpose**: Serves as a container for organizing related controls together.
- **Common Properties**:
    - `Text`: The label text displayed at the top of the `GroupBox`.
    - `Controls`: Collection of controls contained within the `GroupBox`.
    - `Enabled`: Indicates whether the `GroupBox` and its child controls are enabled.
- **Common Methods**:
    - `Focus()`: Sets input focus to the `GroupBox`.

  - `SuspendLayout()` and `ResumeLayout()`: Temporarily suspends and then resumes the layout logic for child controls, improving performance when adding multiple controls.
- **Common Events**:
  - `Enter`: Triggered when the `GroupBox` receives focus.
  - `Leave`: Triggered when the `GroupBox` loses focus.

7. ListView Control

- **Purpose**: Displays a list of items with different views (Details, Large Icon, Small Icon, List, Tile).
- **Common Properties**:
  - `View`: Determines how items are displayed (e.g., Details, LargeIcon, SmallIcon, List, Tile).
  - `Items`: Collection of items displayed in the `ListView`.
  - `Columns`: Collection of columns in the `Details` view.
  - `MultiSelect`: Indicates whether multiple items can be selected.
- **Common Methods**:
  - `AddItem(ListViewItem)`: Adds an item to the `ListView`.
  - `RemoveItem(ListViewItem)`: Removes an item from the `ListView`.
  - `Clear()`: Removes all items from the `ListView`.
- **Common Events**:
  - `SelectedIndexChanged`: Triggered when the selected item in the `ListView` changes.
  - `ItemActivate`: Triggered when an item in the `ListView` is double-clicked.
  - `MouseClick`: Triggered when an item is clicked.

8. TreeView Control

- **Purpose**: Displays a hierarchical collection of labeled items, each of which can have sub-items (nodes).
- **Common Properties**:
  - `Nodes`: Collection of top-level nodes in the `TreeView`.
  - `SelectedNode`: The currently selected node in the `TreeView`.
  - `CheckBoxes`: Indicates whether checkboxes are displayed next to nodes.

    - `ImageList`: Associates images with nodes in the `TreeView`.
- **Common Methods**:
    - `AddNode(TreeNode)`: Adds a node to the `TreeView`.
    - `RemoveNode(TreeNode)`: Removes a node from the `TreeView`.
    - `ExpandAll()`: Expands all nodes in the `TreeView`.
    - `CollapseAll()`: Collapses all nodes in the `TreeView`.
- **Common Events**:
    - `AfterSelect`: Triggered after a node is selected.
    - `NodeMouseClick`: Triggered when a node is clicked with the mouse.
    - `AfterCheck`: Triggered when a node's checked state changes.

9. TabControl

- **Purpose**: Allows users to switch between different pages or tabs, each containing its own set of controls.
- **Common Properties**:
    - `TabPages`: Collection of tabs within the `TabControl`.
    - `SelectedIndex`: The zero-based index of the currently selected tab.
    - `SelectedTab`: The currently selected tab page.
- **Common Methods**:
    - `AddTab(TabPage)`: Adds a new tab to the `TabControl`.
    - `RemoveTab(TabPage)`: Removes a tab from the `TabControl`.
    - `SelectTab(int index)`: Selects the tab at the specified index.
- **Common Events**:
    - `SelectedIndexChanged`: Triggered when the selected tab changes.
    - `Click`: Triggered when a tab is clicked.

10. ContextMenuStrip

- **Purpose**: Provides a context-sensitive menu that appears when the user right-clicks a control.
- **Common Properties**:
    - `Items`: Collection of menu items within the `ContextMenuStrip`.
    - `SourceControl`: The control that is displaying the context menu.

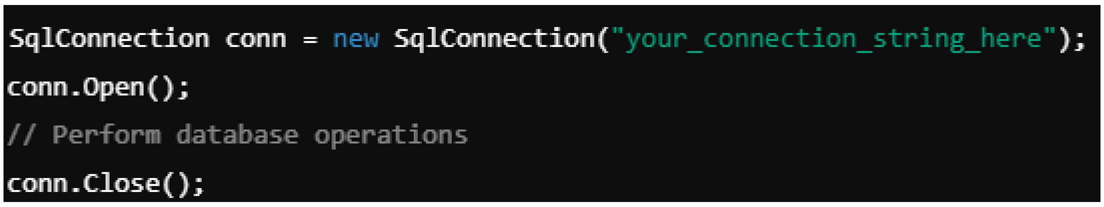
- **Common Methods**:
    - `Show(Control, Point)`: Displays the context menu at the specified location relative to the control.
    - `Hide()`: Hides the context menu.
- **Common Events**:
    - `Opening`: Triggered before the context menu is displayed.
    - `Closing`: Triggered when the context menu is about to close.
    - `ItemClicked`: Triggered when an item in the context menu is clicked.

DATABASE CONNECTIVITY

- **.NET** is a data access technology used in .NET for connecting to databases, retrieving data, and performing CRUD (Create, Read, Update, Delete) operations.
- **Architecture**:
  - **Data Providers**: Components used to connect to a database, execute commands, and retrieve results.
    - **Common Data Providers**:
      - **SQL Server (`System.Data.SqlClient`)**: For SQL Server databases.
      - **OLEDB (`System.Data.OleDb`)**: For databases that support OLEDB.
      - **ODBC (`System.Data.Odbc`)**: For databases that support ODBC.
      - **Oracle (`System.Data.OracleClient`)**: For Oracle databases.

2. Key Components of ADO.NET

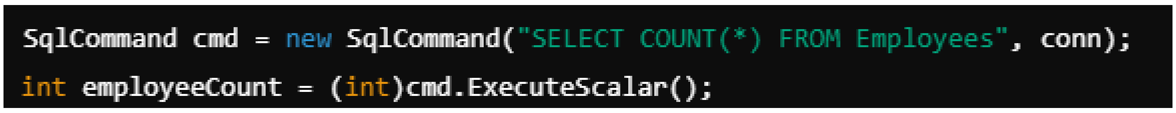
1. **Connection**
   - **Purpose**: Establishes a connection to the database.
   - **Class**: `SqlConnection` (for SQL Server).
   - **Common Properties**:
     - `ConnectionString`: Specifies the database source, credentials, and other connection parameters.
     - `State`: Indicates the current state of the connection (e.g., Open, Closed).
   - **Common Methods**:
     - `Open()`: Opens the connection to the database.
     - `Close()`: Closes the connection.
   - **Example**:



```
SqlConnection conn = new SqlConnection("your_connection_string_here");
conn.Open();
// Perform database operations
conn.Close();
```

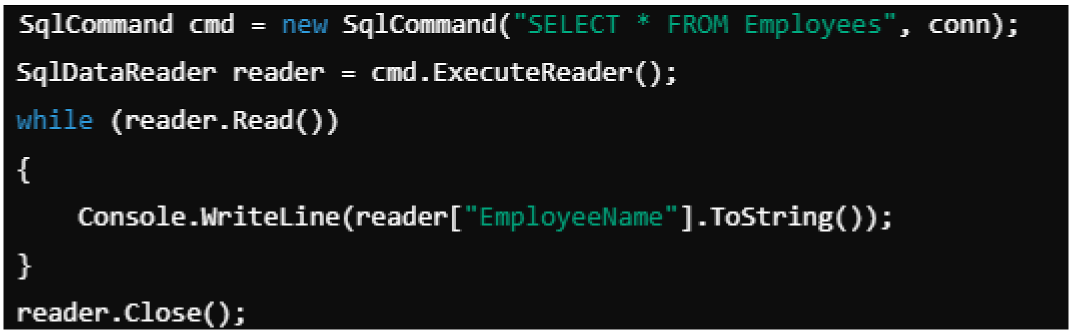
2. **Command**
   - **Purpose**: Executes SQL queries and commands against the database.
   - **Class**: `SqlCommand` (for SQL Server).
   - **Common Properties**:
     - `CommandText`: The SQL query or stored procedure to execute.
     - `CommandType`: Specifies whether the command is a text query, a stored procedure, etc.
     - `Parameters`: Collection of parameters used in the command.
   - **Common Methods**:

        - `ExecuteNonQuery()`: Executes a command that does not return any data (e.g., INSERT, UPDATE, DELETE).
        - `ExecuteScalar()`: Executes a command and returns a single value (e.g., COUNT, SUM).
        - `ExecuteReader()`: Executes a command and returns a `SqlDataReader` to read the data.
    - **Example**:



```
SqlCommand cmd = new SqlCommand("SELECT COUNT(*) FROM Employees", conn);
int employeeCount = (int)cmd.ExecuteScalar();
```

3. **DataReader**
    - **Purpose**: Provides a way to read data from the database in a forward-only, read-only manner.
    - **Class**: `SqlDataReader` (for SQL Server).
    - **Common Properties**:
        - `HasRows`: Indicates whether the `SqlDataReader` contains one or more rows.
        - `FieldCount`: The number of columns in the current row.
    - **Common Methods**:
        - `Read()`: Advances the `SqlDataReader` to the next record.
        - `GetValue(int)`: Retrieves the value of a specified column in its native format.
        - `Close()`: Closes the `SqlDataReader`.
    - **Example**:



```
SqlCommand cmd = new SqlCommand("SELECT * FROM Employees", conn);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    Console.WriteLine(reader["EmployeeName"].ToString());
}
reader.Close();
```

4. **DataAdapter**
    - **Purpose**: Bridges the gap between the `DataSet` and the database for retrieving and saving data.
    - **Class**: `SqlDataAdapter` (for SQL Server).
    - **Common Properties**:
        - `SelectCommand`: The SQL command used to select data.
        - `InsertCommand`: The SQL command used to insert data.
        - `UpdateCommand`: The SQL command used to update data.
        - `DeleteCommand`: The SQL command used to delete data.
    - **Common Methods**:
        - `Fill(DataSet)`: Fills a `DataSet` with data from the database.

    - `Update(DataSet)`: Sends changes made in the `DataSet` back to the database.
  - **Example**:

```
SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM Employees", conn);
DataSet ds = new DataSet();
adapter.Fill(ds, "Employees");
```

5. **DataSet and DataTable**
   - **Purpose**: Represents an in-memory cache of data that can hold multiple tables (`DataSet`) or a single table (`DataTable`).
   - **DataSet**:
     - **Common Properties**:
       - `Tables`: Collection of `DataTable` objects within the `DataSet`.
     - **Common Methods**:
       - `AcceptChanges()`: Commits all the changes made to the `DataSet`.
       - `RejectChanges()`: Rolls back all changes made to the `DataSet`.
     - **Example**:

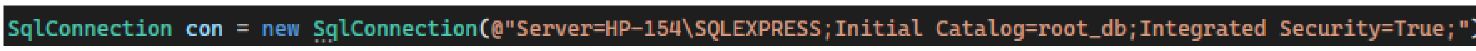
```
DataSet ds = new DataSet();
SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM Employees", conn);
adapter.Fill(ds, "Employees");
```

   - **DataTable**:
     - **Common Properties**:
       - `Rows`: Collection of rows in the `DataTable`.
       - `Columns`: Collection of columns in the `DataTable`.
     - **Common Methods**:
       - `NewRow()`: Creates a new row.
       - `AcceptChanges()`: Commits all the changes made to the `DataTable`.
       - `RejectChanges()`: Rolls back all changes made to the `DataTable`.
     - **Example**:

```
DataTable dt = ds.Tables["Employees"];
foreach (DataRow row in dt.Rows)
{
    Console.WriteLine(row["EmployeeName"].ToString());
}
```

3. Steps to Connect a Windows Forms Application to a Database

1. **Create a Connection String**:
   - The connection string contains information such as the database server, database name, user credentials, and other configuration settings.
   - Example:



```
SqlConnection con = new SqlConnection(@"Server=HP-154\SQLEXPRESS;Initial Catalog=root_db;Integrated Security=True;"
```

2. Establish a Connection:

   - Use `SqlConnection` to establish a connection to the database.
   - Example:

```
using (SqlConnection conn = new qlConnection(connectionString))
{
    conn.Open();


                  Perform database operations


    conn.Close();
}
```

3. Execute Commands:

   - Use `SqlCommand` to execute SQL queries or stored procedures.
   - Example:

```
SqlCommand cmd = new SqlCommand("SELECT * FROM Employees", conn);
SqlDataReader reader = cmd.ExecuteReader();
```

4. Retrieve Data:

   - Use `SqlDataReader` for reading data or `SqlDataAdapter` to fill a `DataSet`.
   - Example:

```
SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM
Employees", conn);
DataSet ds = new DataSet();
adapter.Fill(ds, "Employees");
```

5.Display Data in Controls:

   - Bind data to Windows Forms controls such as `DataGridView`, `ListBox`, `ComboBox`, etc.
   - Example:

```
dataGridView1.DataSource = ds.Tables["Employees"];
```

6. **Insert, Update, and Delete Operations**:
   - Use `SqlCommand` to perform insert, update, and delete operations.
   - Example:

```
SqlCommand insertCmd = new SqlCommand("INSERT INTO Employees
(Name, Age) VALUES (@Name, @Age)", conn);
insertCmd.Parameters.AddWithValue("@Name", "John Doe");
insertCmd.Parameters.AddWithValue("@Age", 30);
insertCmd.ExecuteNonQuery();
```

Handling Database Exceptions

- **Try-Catch Blocks**: Use try-catch blocks to handle exceptions like connection failures, SQL syntax errors, etc.
- **Common Exceptions**:
  - `SqlException`: Represents an error that occurs when interacting with SQL Server.
  - `InvalidOperationException`: Occurs when the operation is not valid for the current state of the connection or command.
- **Example**:

```
try
{
    conn.Open();
    // Perform database operations
}
catch (SqlException ex)
{
    MessageBox.Show("An error occurred: " + ex.Message);
}
finally
{
    conn.Close();
}
```