Microsoft® Official Course

Understanding Desktop Applications

Lesson 5

Microsoft

Objective Domain Matrix

Skills/Concepts	MTA Exam Objectives
Understanding Windows Forms Applications	Understand Windows Forms applications (5.1)
Understanding Console-Based Applications	Understand console-based applications (5.2)
Understanding Windows Services	Understand Windows services (5.3)

Understanding Windows Forms Applications

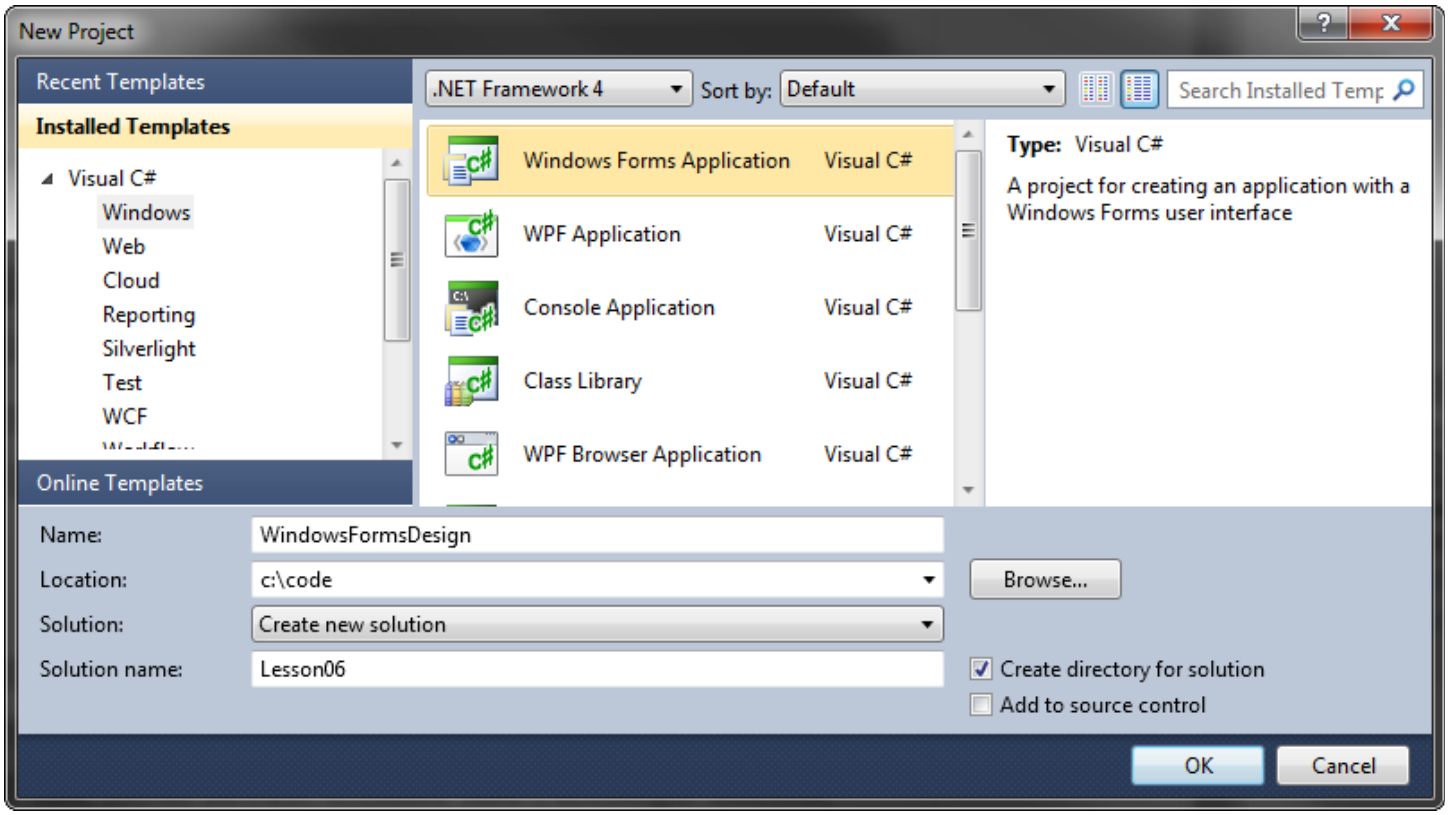
- Windows Forms applications are smart client applications consisting of one or more forms that display a visual interface to the user.
- A Windows Form is a visual surface capable of displaying a variety of controls, including text boxes, buttons, and menus.
- A control is a distinct user interface element that accepts input from the user or displays output to the user.

Designing a Windows Form

- Visual Studio provides a drag-and-drop Windows Forms designer.
- Windows Forms includes a large collection of common controls.
- For specialized interfaces, you can create a custom-control.

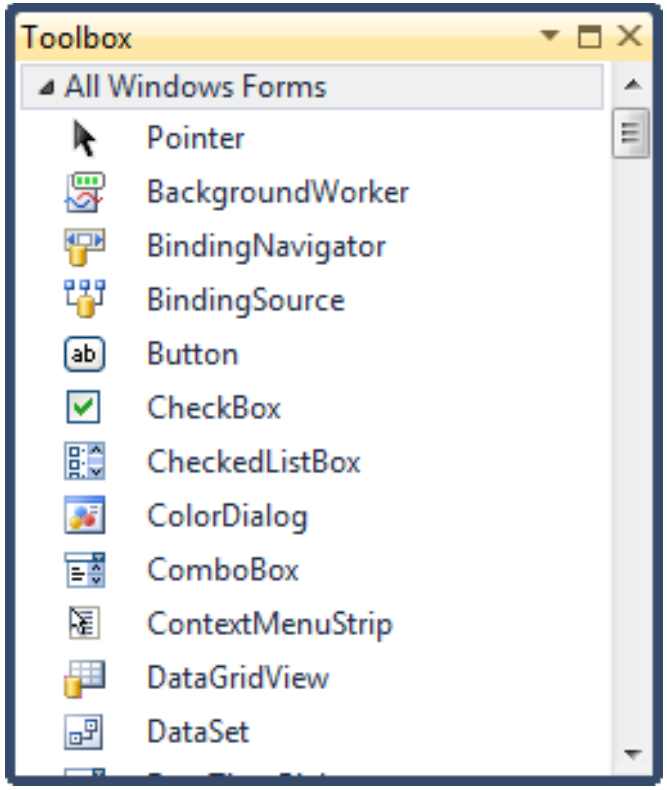
Creating a Windows Form

- Create projects based on the Visual Studio's Windows Forms Application template.



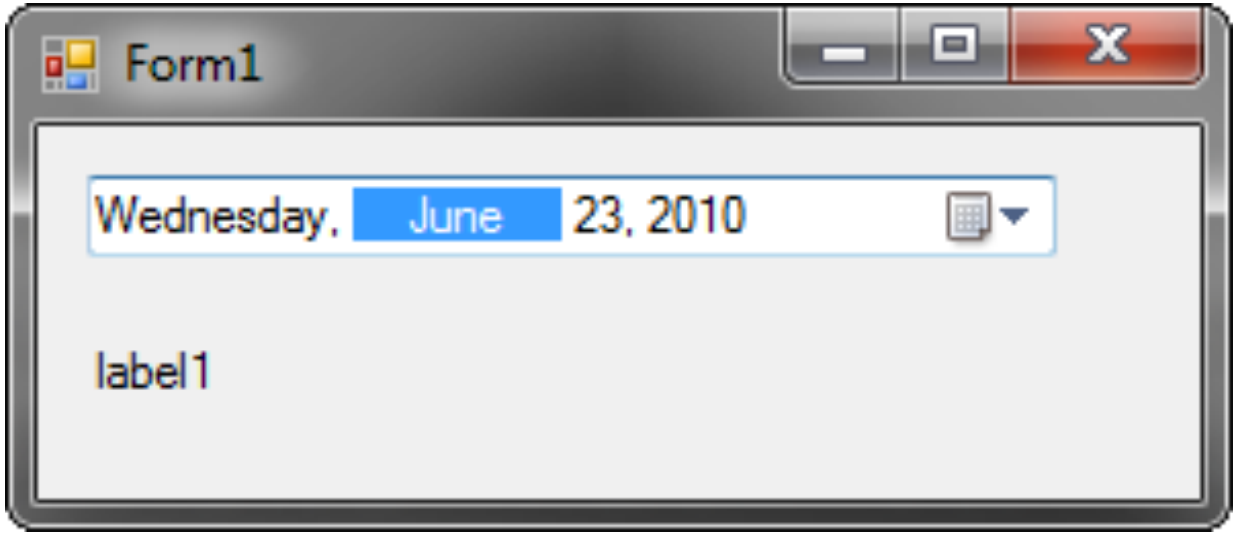
Visual Studio Toolbox

- The Visual Studio toolbox provides common controls.



Windows Forms User Interface

- You can drag and drop controls from the Toolbox and arrange them on the Windows Forms designer to create the user interface.



Windows Forms Event Model

- A form and its components respond to user actions such as keystrokes or mouse movement. These actions are called events.
- Much of the code that you write as a Windows Forms developer is directed toward capturing such events and responding.
- The Windows Forms event model uses .NET Framework delegates to bind events to their respective event handlers.

Handling Events

```
this.dateTimePicker1.ValueChanged +=
    new System.EventHandler(
        this.dateTimePicker1_ValueChanged);
```

- Here, ValueChanged is the event of the DateTimePicker control that we want to capture.
- A new instance of the delegate of type EventHandler is created and the method dateTimePicker1_ValueChanged is passed to the event handler.
- The dateTimePicker1_ValueChanged is the method in which you will write the event-handling code.

Handling Events - Example

```
private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
{
    label1.Text = dateTimePicker1.Value.ToLongDateString();
}
```

- The dateTimePicker1_ValueChanged method is invoked when the ValueChanged event is fired on the dateTimePicker1 control.
- The parameter of the object type specify the object that raised the event.
- The parameter of the EventArgs type provides information about the event.

Visual Inheritance

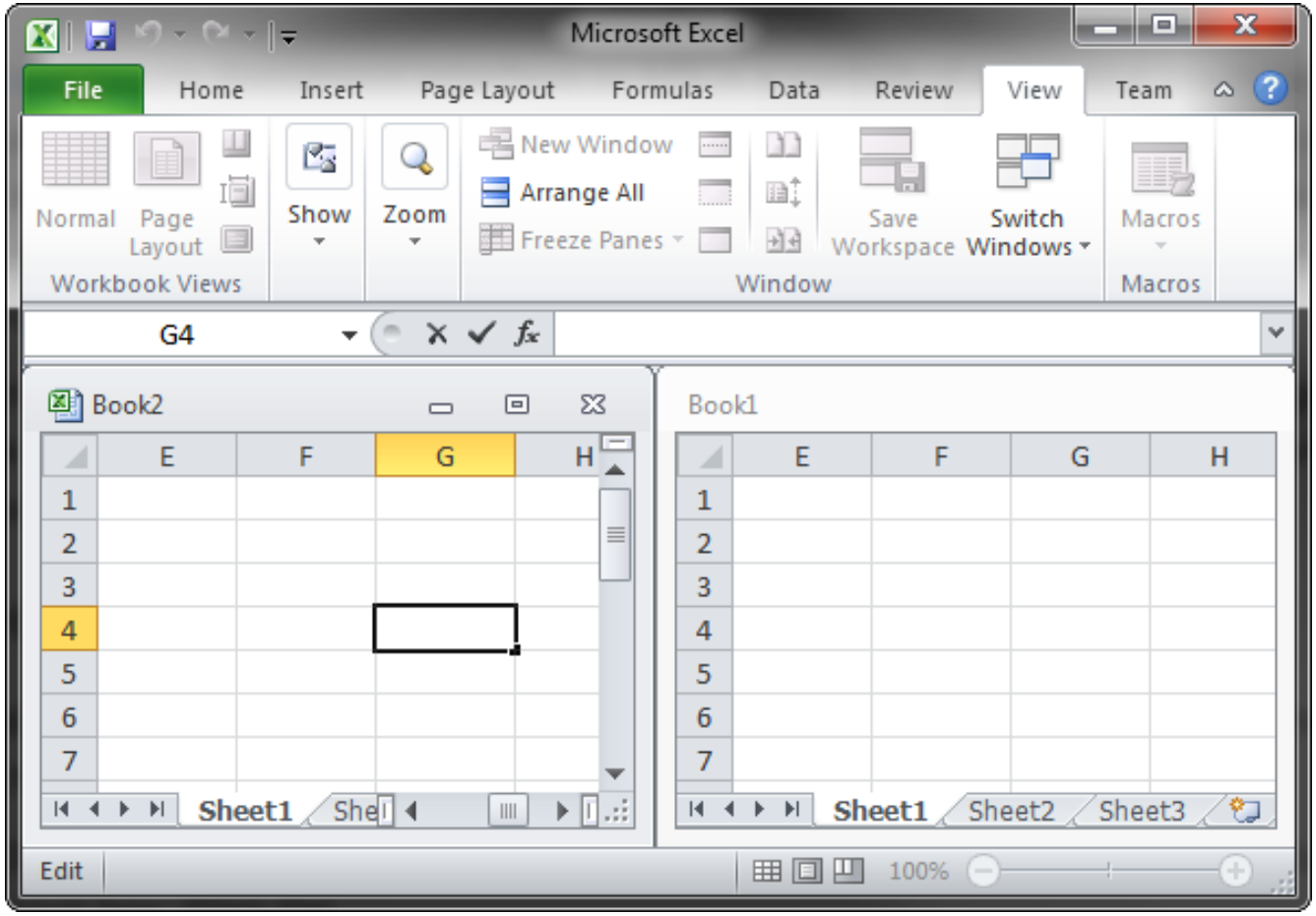
- Visual inheritance allows you to reuse existing functionality and layout for Windows Forms.
- When inheritance is applied to a Windows Form, it causes the inheritance of all the visual characteristics of a form, such as size, color, and any controls placed on the form.
- You can also visually manipulate any of the properties that are inherited from the base class.

```
public partial class InheritedForm : Form1
{
    public InheritedForm()
    {
        InitializeComponent();
    }
}
```

Multiple Document Interface (MDI) Applications

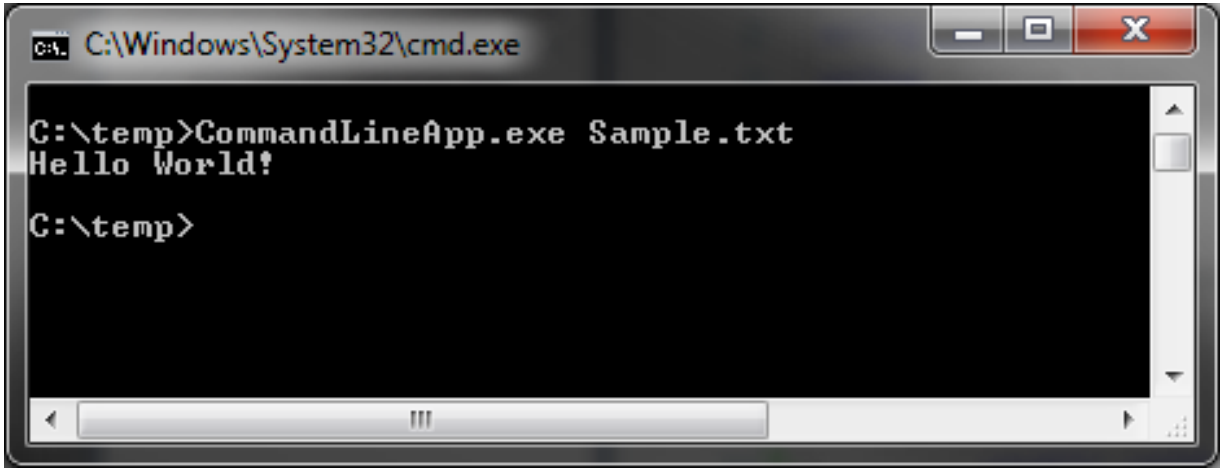
- Multiple Document Interface (MDI) applications are applications in which multiple child windows reside under a single parent window.
- MDI applications allow multiple windows to share a single application menu and toolbar.
- With Single document interface (SDI) applications, each window contains its own menu and toolbar.
- SDI applications rely on the operating system to provide window management functionality.
- To create an MDI application, set the **IsMdiContainer** property to True.
- For the child windows, set the **MdiParent** property to refer to the parent form.

MDI Application Sample



Console-Based Applications

- Console-based applications do not have a graphical user interface and use a text-mode console window to interact with the user.
- Console applications are best suited for tasks that require minimal or no user interface.

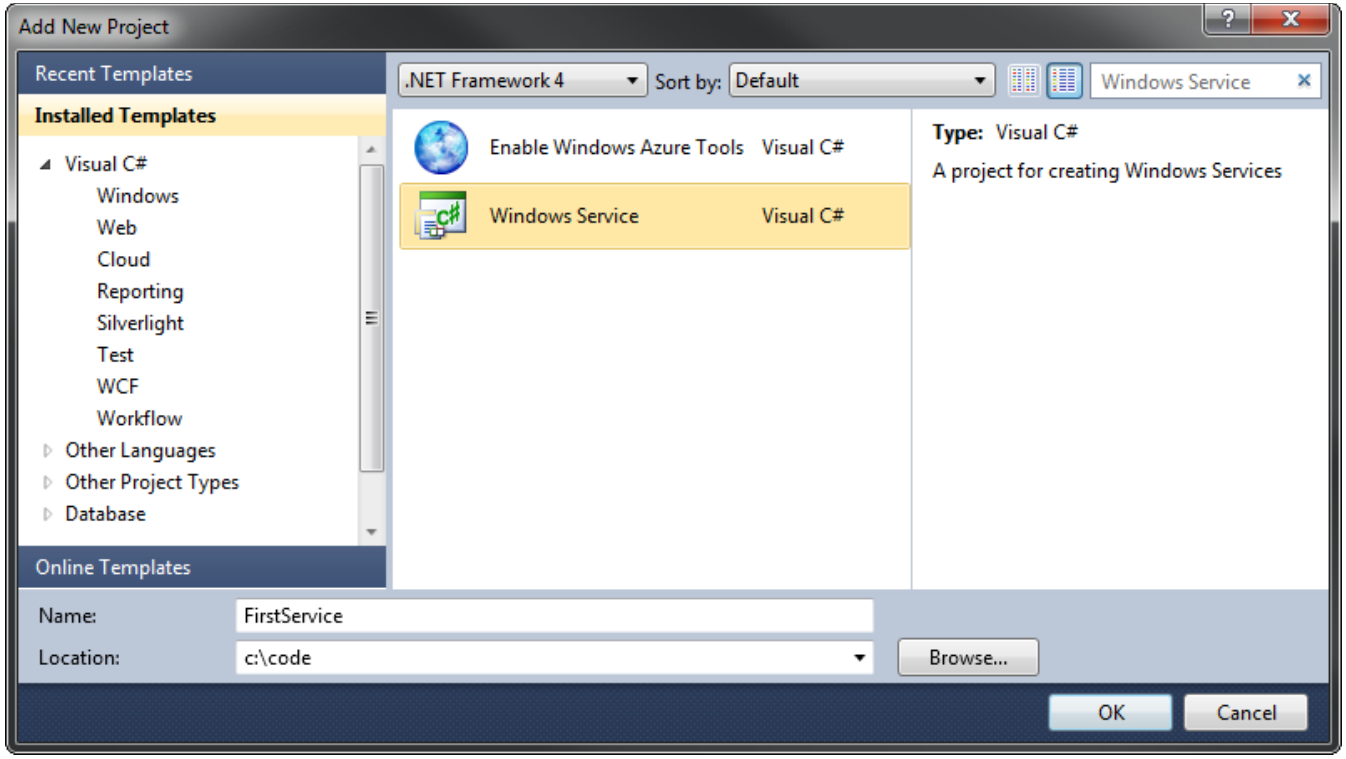


Windows Service Applications

- A Windows service is an application that runs in the background and does not have any user interface.
- Ideal for creating long-running programs that run in the background and do not directly provide any user interaction.
- A Windows service can be started, paused, restarted, and stopped.
- A Windows service can also be set to start automatically when the computer is started.
- Services play an important role in enterprise application architecture. For example, you can have a service that listens for incoming orders and starts an order-processing workflow whenever an order is received.

Creating Windows Service Applications

- Use the Windows Service project template to create Windows Services.



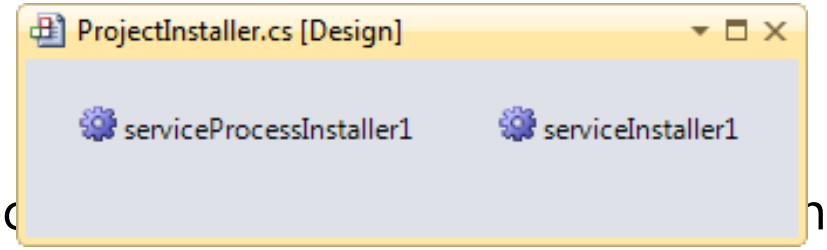
Creating Windows Service Applications

- The Windows Services inherits from the **ServiceBase** class. Override the **OnStart**, **OnStop**, **OnPause**, **OnContinue** and **OnShutdown** method to customize service behavior.

```
protected override void OnStart(string[] args)
{
    eventLog1.WriteEntry("Starting the service", EventLogEntryType.Information, 1001);
}

protected override void OnStop()
{
    eventLog1.WriteEntry("Stopping the service", EventLogEntryType.Information, 1001);
}

protected override void OnPause()
{
    eventLog1.WriteEntry("Pausing the service", EventLogEntryType.Information, 1001);
}

protected override void OnContinue()
{
    eventLog1.WriteEntry("Continuing the service", EventLogEntryType.Information, 1001);
}

protected override void OnShutdown()
{
    eventLog1.WriteEntry("Shutting down the service", EventLogEntryType.Information, 1001);
}
```

Installing Windows Service Applications

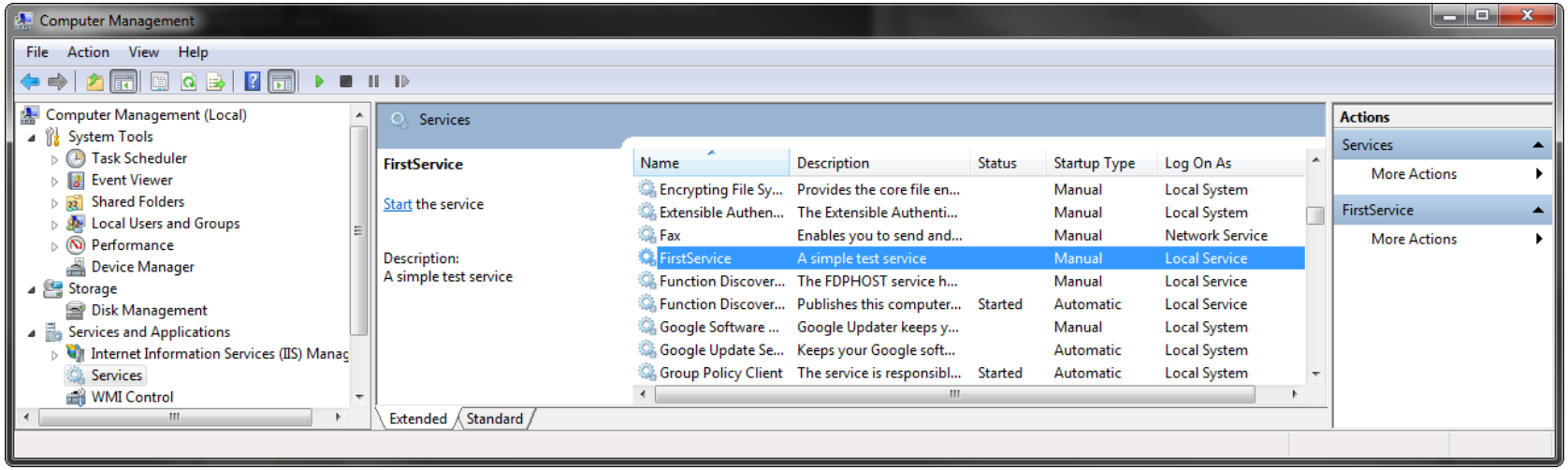
- A Windows Service must be installed before use.
- The **ServiceProcessInstaller** class performs installation tasks that are common to all the Windows services in an application.
  - Setting the login account for the Windows service.
- The **ServiceInstaller** class, on the other hand, performs the installation tasks that are specific to a single Windows service.
  - setting the **ServiceName** and **StartType**.



- An exec ne service installer classes can be installed by using the command line Installer tool (installutil.exe).

Managing Windows Service Applications

- Use the Windows Services tool to start, stop, pause and continue a Windows service.



Recap

- Windows Forms Applications
  - Designing a Windows Form
  - Windows Forms Event Model
  - Visual Inheritance
  - MDI Application vs. SDI Application
- Console-Based Applications
  - Command-line Arguments
- Windows Service Applications
  - Creating a Windows Service