

## Notikumu realizācijas un apstrādes mehānismi .NET ietvarā

- ▶ Notikumu izraisīšanas un apstrādes mehānisms balstās uz delegātu pielietošanu;
- ▶ Delegāta instance specificē vienu vai vairākas atsaucis uz metodēm (t.s. izsaukumu sarakstu), ar konkrētiem parametriem un atgriežamo tipu;
- ▶ Ar delegātu palīdzību, metodes var izmantot kā objektus, kurus var piešķirt mainīgajiem un nodot metožu parametros;
- ▶ Galvenā delegātu priekšrocība ir tā, ka izsaucamās metodes tiek uzstādītas nevis kompilēšanas, bet lietojumprogrammas izpildes laikā;
- ▶ Valodā C++ līdzīgu (bet ne analogisku) funkcionalitāti nodrošina rādītāji uz funkcijām vai abstraktas klases ar virtuālām funkcijām.

## Notikumu realizācijas un apstrādes mehānismi, turp.

```

delegate void SimpleDelegate(int x); // delegāta deklarācijas piemērs

class Test
{
    public void M1(int p) { Console.WriteLine("Test.M1: " + p); }
    public static void M2(int p) { Console.WriteLine("Test.M2: " + p + " (static)"); }
    public void M3(double p) { Console.WriteLine("Test.M3: " + p); }
    public int M4(int p) { Console.WriteLine("Test.M4: " + p); return p; }
}

class Program
{
    static void Main(string[] args)
    {
        Test t = new Test();
        SimpleDelegate d1 = t.M1; // saīsināts pieraksts no: new SimpleDelegate(t.M1);
        SimpleDelegate d2 = Test.M2; // statiska metode
        SimpleDelegate d3 = d1 + d2; // izsaucot delegātu d3(), secīgi tiks izsaukti t.M1() un Test.M2()
        //d3 += t.M3; // kļūda - metodes M3() parametra tips nesakrīt ar delegāta parametra tipu
        //d3 += t.M4; // kļūda - metodes M4() rezultāta tips nesakrīt ar delegāta rezultāta tipu

        d1(1); // tiek izsaukta metode t.M1(1);
        d2(2); // tiek izsaukta metode Test.M2(2);
        d3(3); // secīgi tiek izsauktas metodes t.M1(3) un Test.M2(3)
        Console.ReadLine();
    }
}

```

```

Test.M1: 1
Test.M2: 2 (static)
Test.M1: 3
Test.M2: 3 (static)

```

## Notikumu realizācijas un apstrādes mehānismi, turp.

- ▶ Deklarējot delegātu, automātiski tiek izveidota klases `System.Delegate` apakšklase, tomēr mantot no delegāta klases nav atļauts;
- ▶ Izsaukuma saraksta papildināšanu veic ar klases `Delegate` operācijām `+`, `+=` vai metodi `Combine()`;
- ▶ Metodes izņemšanu no izsaukuma saraksta veic ar klases `Delegate` operācijām `-`, `-=` vai metodi `Remove()`;
- ▶ Delegāta instance vienmēr norāda uz vienu un to pašu mērķa objektu un metodi, vai klases metodi;
- ▶ Ja ar operācijām tiek apvienoti vairāki delegāti, tiek izveidota jauna delegāta instance, ar savu izsaukumu sarakstu, kas ir apvienoto delegātu izsaukumu sarakstu secīgs kombinējums.

## Notikumu realizācijas un apstrādes mehānismi, turp.

- ▶ Izsaucot delegātu:
- ▶ Ja izsaukumu sarakstā ir tikai viens elements, tiek izsaukta tā metode, parametrā norādot delegāta instances parametrus, un tiek atgriezta metodes vērtība;
- ▶ Ja izsaukuma sarakstā ir vairāki elementi, to metodes tiek izsauktas secīgi viena aiz otras, katrai parametrā norādot vienus un tos pašus delegāta instances parametrus, un tiek atgriezta pēdējās izsauktās metodes vērtība;
- ▶ Ja izsaukumu secībā, kāda metode izraisa izņēmumu, kas netiek apstrādāts, atlikušās izsaukuma sarakstā esošās metodes netiek izsauktas, jo notiek atkāpšanās uz atbilstošu izņēmuma apstrādātāju;
- ▶ Ja delegāta izsaukuma sarakstā nav neviena elements, izsaucot delegātu, tiek izraisīts izņēmums ar tipu `System.NullReferenceException`.

## Notikumu realizācijas un apstrādes mehānismi, turp.

- ▶ Anonīmās metodes kā delegātu izsaukuma objekti:

```
delegate int SumIt(int upto);

class Program
{
    static void Main(string[] args)
    {
        SumIt sumit1 = delegate(int upto)
        {
            int sum = 0;
            for (int i = 1; i <= upto; i++)
                sum += i;
            return sum;
        };
        Console.WriteLine("Sum up to 5 = " + sumit1(5));
        Console.WriteLine("Sum up to 10 = " + sumit1(10));
        Console.ReadLine();
    }
}
```

} Anonīmā metode

## Notikumu realizācijas un apstrādes mehānismi, turp.

- ▶ Galvenie delegātu pielietojanas mērķi:
- ▶ Iespēja noteikt izsaukamo metodi, nevis lietojumprogrammas kompilēšanas laikā, bet gan tās izpildes laikā;
- ▶ Saites “avots – novērotājs” realizēšanai starp objektiem;
- ▶ Universālu metožu izveidošanai, kuras var nodot kā parametrus citām metodēm;
- ▶ Atpakaļizsaukšanas (callback) mehānisma nodrošināšanai.

## Notikumu realizācijas un apstrādes mehānismi, turp.

- ▶ **Notikums:**
  - automātiska izziņošana par noteiktas darbības veikšanu vai stāvokļa iestāšanos;
  - C# klases elements, caur kuru tiek paziņots citiem objektiem par objekta stāvokļa maiņu;
  - cieši saistīts ar delegātu mehānismu;
- ▶ **Notikuma izveidošana ietver šādus soļus:**
  - delegāta, kas apraksta notikuma apstrādātāju signatūru, deklarēšana;
  - notikuma kā klases elementa deklarēšana;
  - metodes, kurā notiek notikuma izraisīšana, definēšana.
- ▶ **Notikuma apstrāde ietver šādus soļus:**
  - notikuma apstrādātāja definēšana klasē(s);
  - notikuma apstrādātāja reģistrēšana objekta notikuma elementā.

Vizuālā programmēšana (studiju projekts)

41

## Notikumu realizācijas un apstrādes mehānismi, turp.

```

delegate int MyEventHandler(int p); // notikuma delegāts

class ClassWithEvent {
    public event MyEventHandler MyEvent; // notikuma elementa deklarēšana
    public void FireEvent() {
        if (MyEvent != null) { // jāpārbauda obligāti!
            Console.WriteLine("Firing event MyEvent().");
            int result = MyEvent(5); // piesaistīto notikuma apstrādātāju izsaukšana
            Console.WriteLine("Result from last event handler = " + result);
        }
    }
}

class Program {
    static void Main(string[] args) {
        ClassWithEvent ce = new ClassWithEvent();
        ce.MyEvent += delegate(int p) {
            Console.WriteLine("Event handler 1 called. p = " + p);
            return p * 2;
        };
        ce.MyEvent += delegate(int q) {
            Console.WriteLine("Event handler 2 called. q = " + q);
            return q * 3;
        };
        ce.FireEvent();
        Console.ReadLine();
    }
}

```

```

Firing event MyEvent().
Event handler 1 called. p = 5
Event handler 2 called. q = 5
Result from last event handler = 15

```

Vizuālā programmēšana (studiju projekts)

42

## Notikumu realizācijas un apstrādes mehānismi, turp.

- ▶ Notikuma elements ir abstrakcija, kas faktiski sastāv no izolētas klases, kurā deklarēts delegāta eksemplārs un divas metodes – notikumu apstrādātāju metožu reģistrēšanai un atreģistrēšanai;
- ▶ Notikuma apstrādātāju reģistrē ar operāciju  
+=
- ▶ Notikuma apstrādātāju atreģistrē ar operāciju  
-=

## Notikumu realizācijas un apstrādes mehānismi, turp.

- ▶ .NET ietvars definē lielu skaitu delegātus, kas paredzēti notikumu mehānisma realizēšanai;
- ▶ Vairums standarta delegātu ir deklarēti pēc vienas shēmas:
  - delegāta vārds beidzas ar piedēkli **EventHandler**;
  - delegāts saņem divus parametrus;
  - pirmais parametrs ir ar tipu **object**, un paredzēts notikuma izraisītāja instances norādīšanai;
  - otrais parametrs definē notikuma parametru, un ir ar tipu **EventArgs** vai kādu tā apakštipu;
  - caur notikuma parametru tiek nodota papildus informācija notikuma apstrādātājam, kā arī notikuma apstrādātājs caur šo parametru var atgriezt informāciju notikuma izraisītājam;
  - delegāta rezultāta tips ir **void**.

## Notikumu realizācijas un apstrādes mehānismi, turp.

```
public delegate void EventHandler(object sender,
    System.EventArgs e);
```

- ▶ Notikuma izraisīšana ir līdzvērtīga metodes `Invoke()` izsaukšanai, piem.:

```
public event EventHandler EventName;
public void FireEvent()
{
    if (EventName != null)
        EventName.Invoke(this, new EventArgs());
}
```

- ▶ Notikuma metodes `BeginInvoke()` un `EndInvoke()` ļauj veikt asinhronu notikumu apstrādāšanu, jo izmanto jaunu pavedienu.

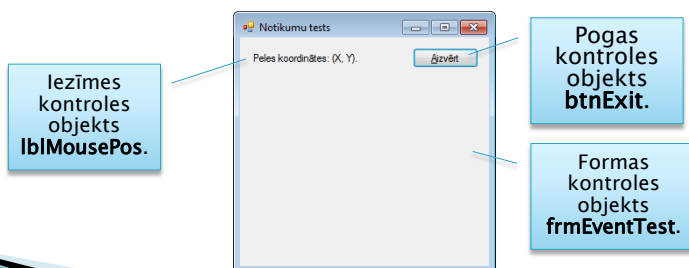
## Lietojumprogrammas dīkstāves notikums Idle

```
void Idle(object sender, EventArgs e)
{
    mnuCopy.Enabled = txtContent.SelectionLength > 0;
    // ...
}
```

```
private void frmTestForm_Load(object sender
    , EventArgs e)
{
    Application.Idle += new EventHandler(Idle);
}
```

## Notikumu apstrādātāju sasaite formas redaktorā

- Formu reaktors rūpējas par to, lai visas uz formas uzvietotās kontroles tiktu dinamiski izveidotas, to īpašības uzstādītas un notikumu apstrādātāji piesaistīti.
- Iepriekš aplūkots piemērs:



Vizuālā programmēšana (studiju projekts)

47

## Notikumu apstrādātāju sasaite formas redaktorā, turp.

- Faila `frmEventTest.cs` saturs:

```
using ...

namespace EventTest
{
    public partial class frmEventTest : Form
    {
        public frmEventTest()
        {
            InitializeComponent();
        }

        private void frmEventTest_MouseMove(object sender, MouseEventArgs e) ...
        private void btnExit_Click(object sender, EventArgs e) ...
        private void frmEventTest_FormClosing(object sender, FormClosingEventArgs e) ...
    }
}
```

Vizuālā programmēšana (studiju projekts)

48

## Notikumu apstrādātāju sasaite formas redaktorā, turp.

- Faila **frmEventTest.Designer.cs** saturs:

```
namespace EventTest {
    partial class frmEventTest {
        // ...
        private void InitializeComponent() {
            this.btnExit = new System.Windows.Forms.Button();
            this.lblMousePos = new System.Windows.Forms.Label();
            // ...
            // btnExit
            this.btnExit.Location = new System.Drawing.Point(197, 12);
            this.btnExit.Name = "btnExit";
            this.btnExit.Size = new System.Drawing.Size(75, 23);
            this.btnExit.TabIndex = 0;
            this.btnExit.Text = "&Aizvērt";
            this.btnExit.UseVisualStyleBackColor = true;
            this.btnExit.Click +=
                new System.EventHandler(this.btnExit_Click);
        }
        // ...
    }
}
```

Vizuālā programmēšana (studiju projekts)

49

## Notikumu apstrādātāju sasaite formas redaktorā, turp.

- Faila **frmEventTest.Designer.cs** saturs, turp.:

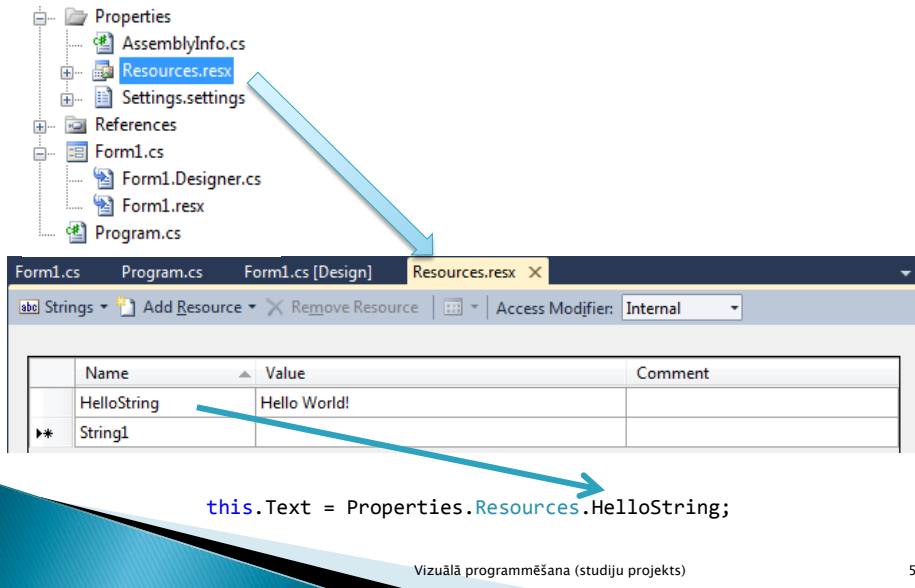
```
namespace EventTest {
    partial class frmEventTest {
        // ...
        private void InitializeComponent() {
            // ...
            this.Controls.Add(this.lblMousePos);
            this.Controls.Add(this.btnExit);
            // ...
            this.FormClosing += new
                System.Windows.Forms.FormClosingEventHandler(this.frmEventTest_FormClosing);
            this.MouseMove += new
                System.Windows.Forms.MouseEventHandler(this.frmEventTest_MouseMove);
            // ...
        }
        // ...
        private System.Windows.Forms.Button btnExit;
        private System.Windows.Forms.Label lblMousePos;
    }
}
```

Vizuālā programmēšana (studiju projekts)

50



## Lietojumprogrammas resursi



Properties

- AssemblyInfo.cs
- Resources.resx
- Settings.settings

References

- Form1.cs
- Form1.Designer.cs
- Form1.resx
- Program.cs

Form1.cs Program.cs Form1.cs [Design] Resources.resx X

Strings Add Resource Remove Resource Access Modifier: Internal

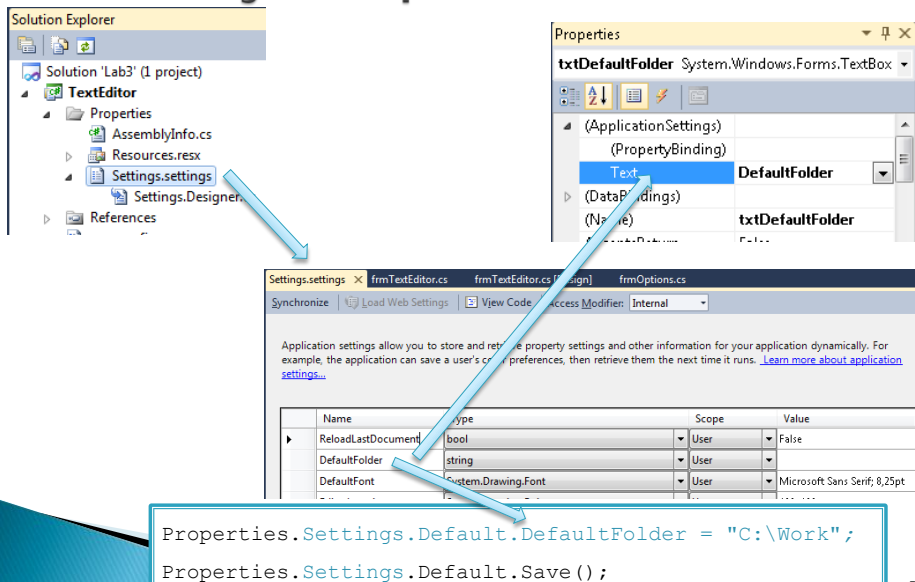
| Name        | Value        | Comment |
|-------------|--------------|---------|
| HelloString | Hello World! |         |
| String1     |              |         |

```
this.Text = Properties.Resources.HelloString;
```

Vizuālā programmēšana (studiju projekts)

51

## Uzstādījumu pārvaldība



Solution Explorer

Solution 'Lab3' (1 project)

- TextEditor
  - Properties
    - AssemblyInfo.cs
    - Resources.resx
    - Settings.settings
    - Settings.Designer.cs
  - References

Properties

txtDefaultFolder System.Windows.Forms.TextBox

(ApplicationSettings) (PropertyBinding)

Text DefaultFolder

(DataBindings) (None) txtDefaultFolder

Settings.settings X frmTextEditor.cs frmTextEditor.cs [Design] frmOptions.cs

Synchronize Load Web Settings View Code Access Modifier: Internal

Application settings allow you to store and retrieve property settings and other information for your application dynamically. For example, the application can save a user's preferences, then retrieve them the next time it runs. [Learn more about application settings.](#)

| Name               | Type                | Scope | Value                        |
|--------------------|---------------------|-------|------------------------------|
| ReloadLastDocument | bool                | User  | False                        |
| DefaultFolder      | string              | User  | C:\Work                      |
| DefaultFont        | System.Drawing.Font | User  | Microsoft Sans Serif, 8.25pt |
| DefaultFontSize    | int                 | User  | 10                           |

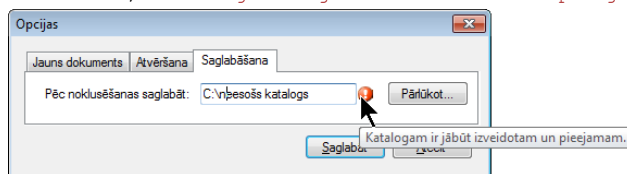
```
Properties.Settings.Default.DefaultFolder = "C:\Work";
Properties.Settings.Default.Save();
```

52

# Komponents ErrorProvider

- ▶ Ļauj vienkāršot ziņošanu par kļūdainu datu apstrādi formas vizuālajos komponentos;
- ▶ Labāka alternatīva nekā ziņojumu logu izvade;
- ▶ Izmantošana:
  - Formas kontroles notikuma **Validating** apstrādātājā veic kontroles datu validāciju un, ja tiek konstatēts, ka dati nav korekti, aktivizē kļūdas situāciju, piem., ar šādu pirmkodu:

```
errorProvider.SetError(txtDefaultFolder
, "Katalogam ir jābūt izveidotam un pieejamam.");
```



- Kļūdas situāciju atceļ, uzstādot tās tekstu ar tukšu virkni:

```
errorProvider.SetError(txtDefaultFolder, "");
```