



## СЪДЪРЖАНИЕ

### I. Йерархия на класовете за форми, прозорци и диалози.

1. Основни класове в Windows Forms за работа с графични компоненти
2. Йерархия на класовете
3. Класът Control
4. Класът ScrollableControl
5. Класът ContainerControl
6. Форми, прозорци и диалози

### II. Управление на събитията.

### III. Използване на диалогови кутии.

1. Стандартни диалогови кутии

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



## 1. Основни класове в Windows Forms за работа с графични компоненти

Библиотеката Windows Forms дефинира съвкупност от базови класове за контролите, контейнер-контролите, както и множество графични контроли и неграфични компоненти.

Основните базови класове, използвани в Windows Forms, са:

- **System.ComponentModel.Component** – представлява .NET компонент. Използва се за реализацията на неграфични компоненти. Например компонентата `System.Windows.Forms.Timer` е наследник на класа `Component`.
- **System.Windows.Forms.Control** – представлява графична контрола. Графични контроли са компонентите, които имат графичен образ. Всички Windows Forms контроли са наследници на класа `Control` включително и контейнер-контролите.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ

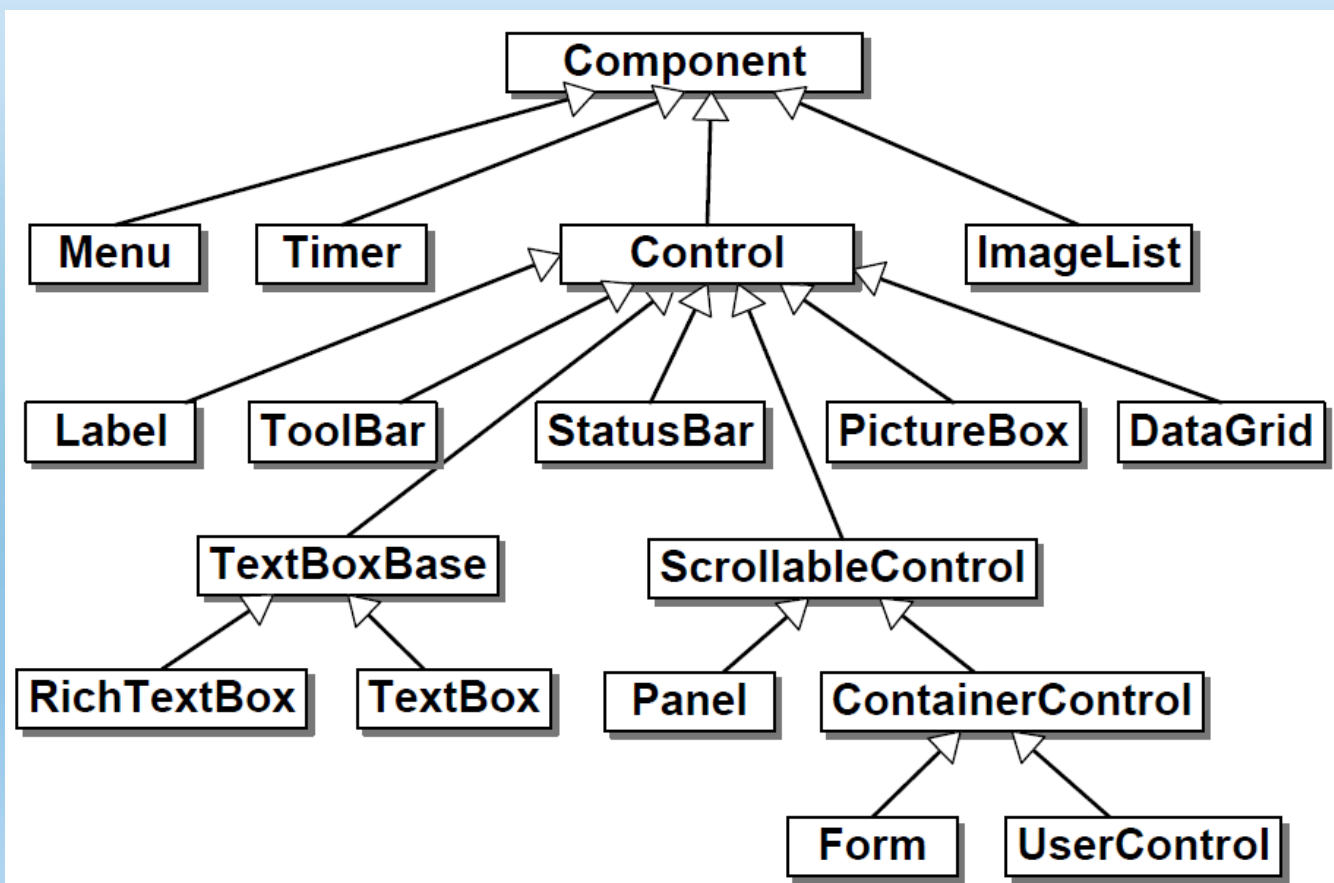


- **System.Windows.Forms.ScrollableControl** – представлява контрола, която поддържа скролиране на съдържанието си. Може да съдържа в себе си други контроли.
- **System.Windows.Forms.ContainerControl** – представлява контрола, която съдържа в себе си други контроли и **осигурява управление на фокуса**. Не всички контейнер-контроли наследяват този клас. Например панелът (System.Windows.Forms.Panel) може да съдържа в себе си други контроли, но е наследник на класа ScrollableControl, а не на ContainerControl.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ

## 2. Йерархия на класовете

На класдиаграмата по-долу е показана част от класовата йерархия на библиотеката Windows Forms:



Не всички класове от Windows Forms са контроли. Някои са обикновени .NET компоненти, например Menu, Timer и ImageList. Менюто не е контрола, но това е така, защото компонентата Menu реално няма графичен образ и представлява списък от MenuItem елементи. MenuItem класът вече има графичен образ и следователно е контрола.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



Типичните контроли (Label, TextBox, Button, ToolBar, StatusBar и др.) са наследници на класа **Control**. Общото за всички тях е, че имат графичен образ и се управляват чрез съобщения.

Контролите, които могат да се скролират (например панелите) са наследници на **ScrollableControl**. Контролите, които съдържат други контроли и се грижат за управление на фокуса (например формите и диалозите), наследяват **ContainerControl**.


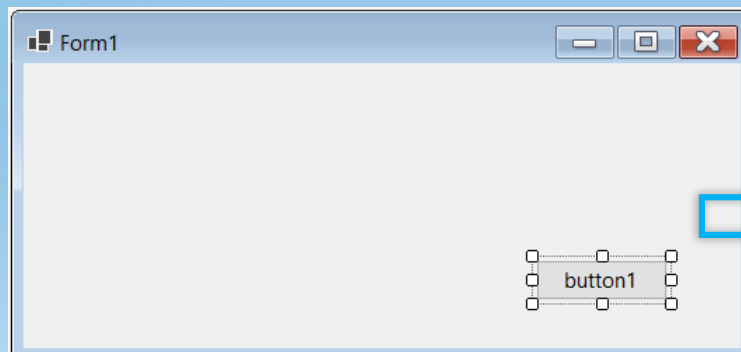
# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



## 3. Класът Control

### 3.1 По-важните свойства на класа Control:

- ❑ **Anchor, Dock** – задават по какъв начин контролата се "закотвя" за контейнера си. Тези свойства са много полезни, ако искаме да управляваме размерите и позицията на контролата при промяна на размерите на контейнера, в който е поставена. Например чрез свойството Anchor можем да закотвим дадена контрола на определено разстояние от долния десен ъгъл на Формата, в която стои, и при преоразмеряване това разстояние ще се запазва и контролата ще се движи заедно с движението на долния десен ъгъл на контейнера, в който е поставена.



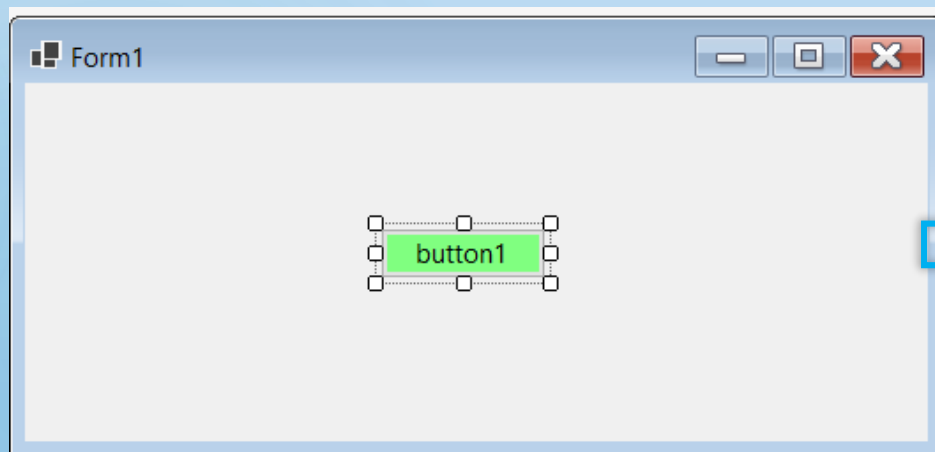
| Layout       |                      |
|--------------|----------------------|
| Anchor       | <u>Bottom, Right</u> |
| AutoSize     | False                |
| AutoSizeMode | GrowOnly             |
| Dock         | None                 |
| Location     | <u>365; 140</u>      |
| Margin       | <u>3; 3; 3; 3</u>    |





# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **BackColor** – задава цвета на фона. Цветовете са инстанции на структурата `System.Drawing.Color`, която дефинира множество стандартни цветове и позволява потребителски дефинирани цветове, състоящи се от 4 на брой 8-битови компонента (яркост, червено, зелено и синьо).

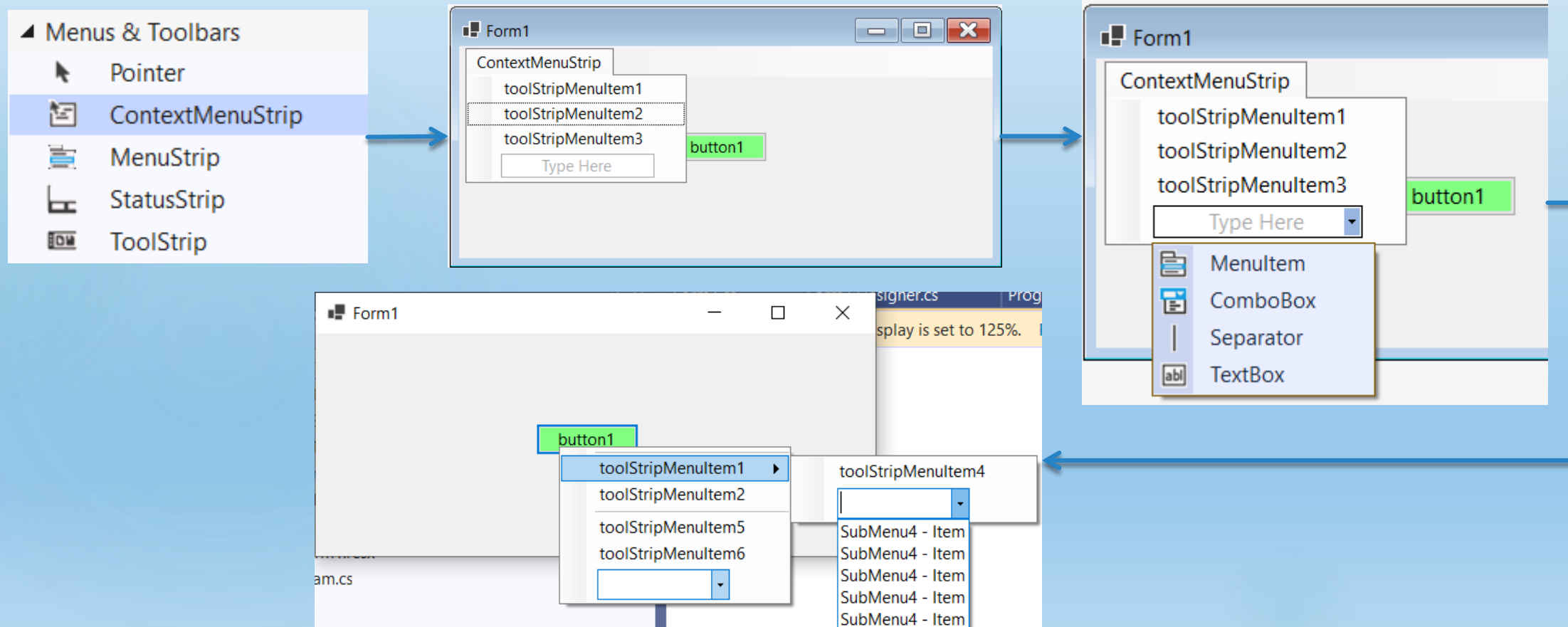


| Appearance            |  |
|-----------------------|--|
| <u>BackColor</u>      |  <b>128; 255; 128</b> |
| BackgroundImage       |  (none)               |
| BackgroundImageLayout | Tile   |
| Cursor                | Default  |

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **ContextMenuStrip** – задава контекстно меню (popup menu) за контролата. Контекстното меню обикновено се появява при натискане на десния бутон на мишката върху контролата.





# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ

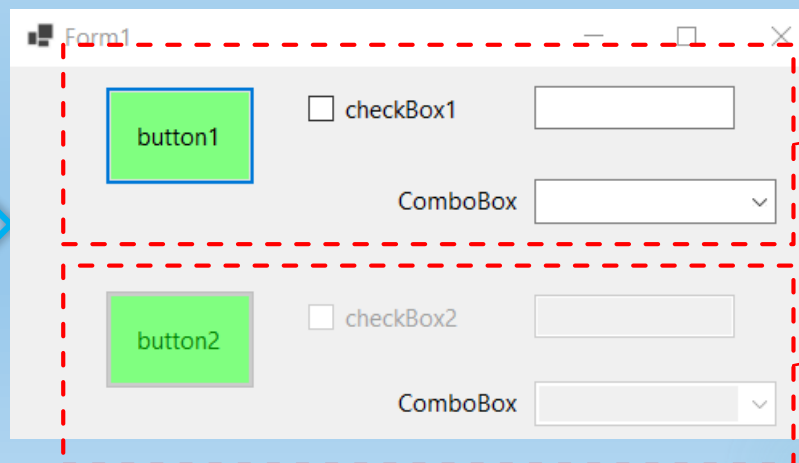
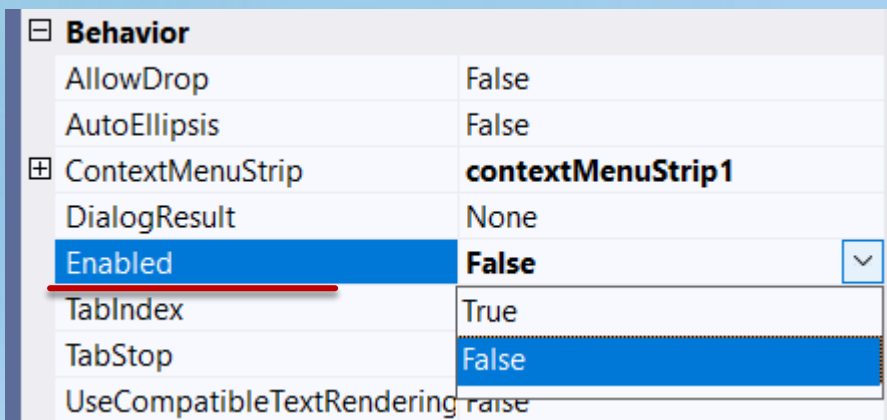


- ❑ **Controls** – съдържа колекция от вложените в контролата други контроли (ако има такива). Например формите (инстанции на класа Form) съдържат в колекцията си Controls контролите, които са разположени в тях. По принцип всички Windows Forms контроли имат колекция Controls и могат да съхраняват в нея други контроли, но за някои от тях не е коректно това да се прави. Например не е коректно в бутон да поставяме друг бутон или текстово поле. Ако го направим, се появяват неприятни аномалии.
- ❑ **CanFocus** – връща дали контролата може да получава фокуса. Почти всички видове контроли могат да бъдат фокусирани, стига да не са забранени (Enabled=false).

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **Enabled** – позволява забраняване на контролата. Когато една контрола бъде забранена (`Enabled=false`), тя остава видима, но става неактивна. Обикновено забранените контроли се изобразяват с избледнял цвят, за да се различават от останалите. Забранените контроли не могат да получават фокуса. В частност забранен бутон не може да бъде натиснат, в забранено текстово поле не може да се пише и т.н. Ако забраним контейнер-контрола, която съдържа в себе си други контроли, всички тези контроли стават забранени.



Активни контроли

Неактивни контроли

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ

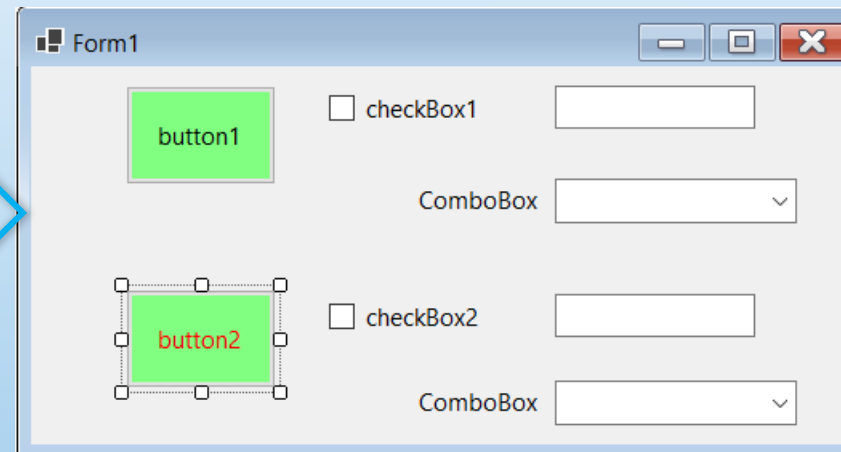
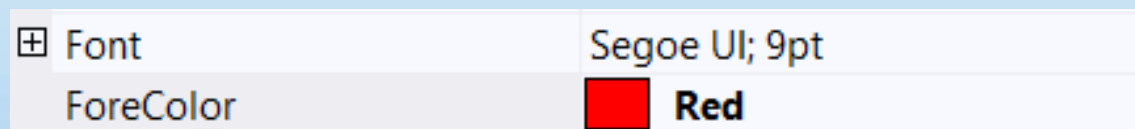


□ **Font** – задава шрифта, с който се изписва текстът в контролата (ако контролата по някакъв начин визуализира текст). При текстови полета това е шрифтът на текста в полето. При бутон това е шрифтът на текста в бутона. При етикет това е шрифтът на текста на етикета. Ако се зададе свойството Font за *Формата*, всички контроли, които не дефинират изрично Font, го наследяват от формата. Шрифтът, с който е изобразено заглавието на *Формите*, не може да се променя от Windows Forms. Той се настройва от графичната среда на операционната система (от контролния панел при Windows). Шрифтовете имат следните характеристики: наименование на шрифт (например Arial) или фамилия шрифтове (например Monospace, SansSerif или Serif), стил (например Bold, Italic, ...), размер (например 12 pt или 10 px) и кодова таблица (Cyrillic, Western, Greek, ...). Кодовата таблица е необходима рядко – само за старите шрифтове, които не поддържат Unicode.

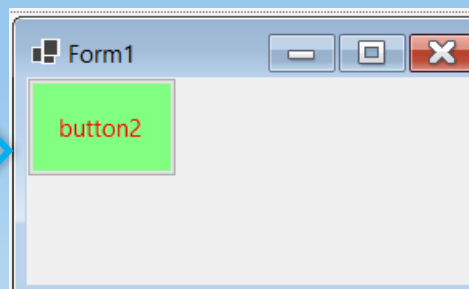
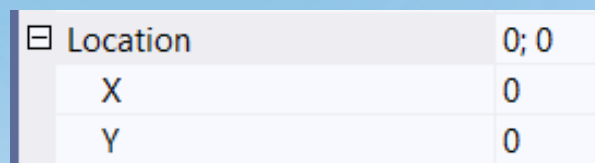
# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **ForeColor** – задава цвета на контролата ( в случая с бутоните това се явява цвета на текста).



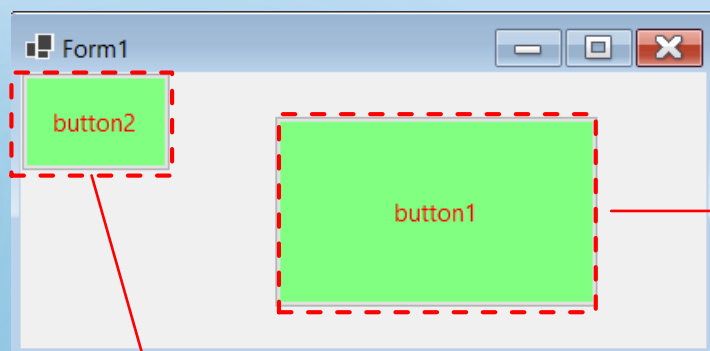
- ❑ **Location** – съдържа позицията на контрола в нейния контейнер (координатите на горния ъгъл). За Форми това е позицията на екрана, а за други контроли това е позицията във *Формата* или контейнер-контролата.



# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **Parent** – задава контейнер-контролата, в която се намира текущата контрола. Може и да няма такава (стойност null). Формите най-често имат стойност null за свойството Parent.
- ❑ **Size** – съдържа размерите на контролата (ширина и височина) **в Пиксели**.



|        |          |
|--------|----------|
| Size   | 203; 120 |
| Width  | 203      |
| Height | 120      |

|        |        |
|--------|--------|
| Size   | 94; 62 |
| Width  | 94     |
| Height | 62     |

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **TabIndex** – **определя реда при навигация** с [Tab] и [Shift+Tab].
- ❑ **TabStop** – задава дали контролата трябва да се фокусира при навигация с [Tab] и [Shift+Tab]. **Ако се зададе TabStop=false, фокусът не спира в контролата при преминаване към следващата контрола (контролата се прескача).**
- ❑ **Text** – **задава текст, свързан с контролата.** При етикет това е текстът, изобразен в етикета. При бутон това е текстът, изобразен в бутона. При текстово поле това е текстът, въведен в полето. При Форма това е заглавието на *Формата*. Текстът е в Unicode и това позволява да се използват свободно букви и знаци на латиница, кирилица, гръцки, арабски и други азбуки, стига избраният шрифт да съдържа съответните знаци.
- ❑ **Visible** – **задава видимост на контролата.** Ако за дадена контрола се зададе Visible=false, тя се скрива (изчезва, все едно не съществува). Скрита контрола може да се покаже отново, като ѝ се зададе Visible=true.



# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



## 3.2 Методи на класа Control:

Публичните методи на класа Control се наследяват и са достъпни във всички Windows Forms контроли. По-важните от тях са:

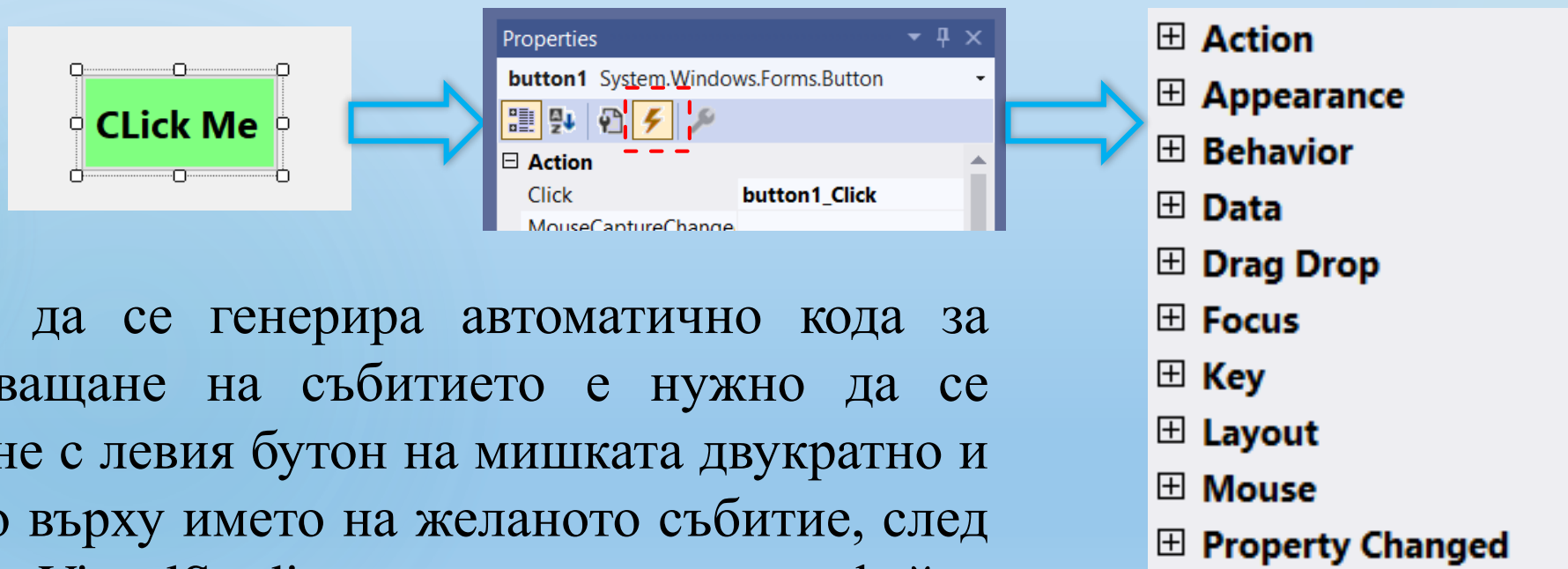
- ❖ **Focus()** – фокусира контролата (ако е възможно).
- ❖ **Hide(), Show()** – скрива/показва контролата (ефектът е като да зададем Visible=false / Visible=true).

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



**3.3 Събития на класа Control** - Благодарение на тях програмистът може да пише код, който се задейства при различни промени в състоянието на контролите.

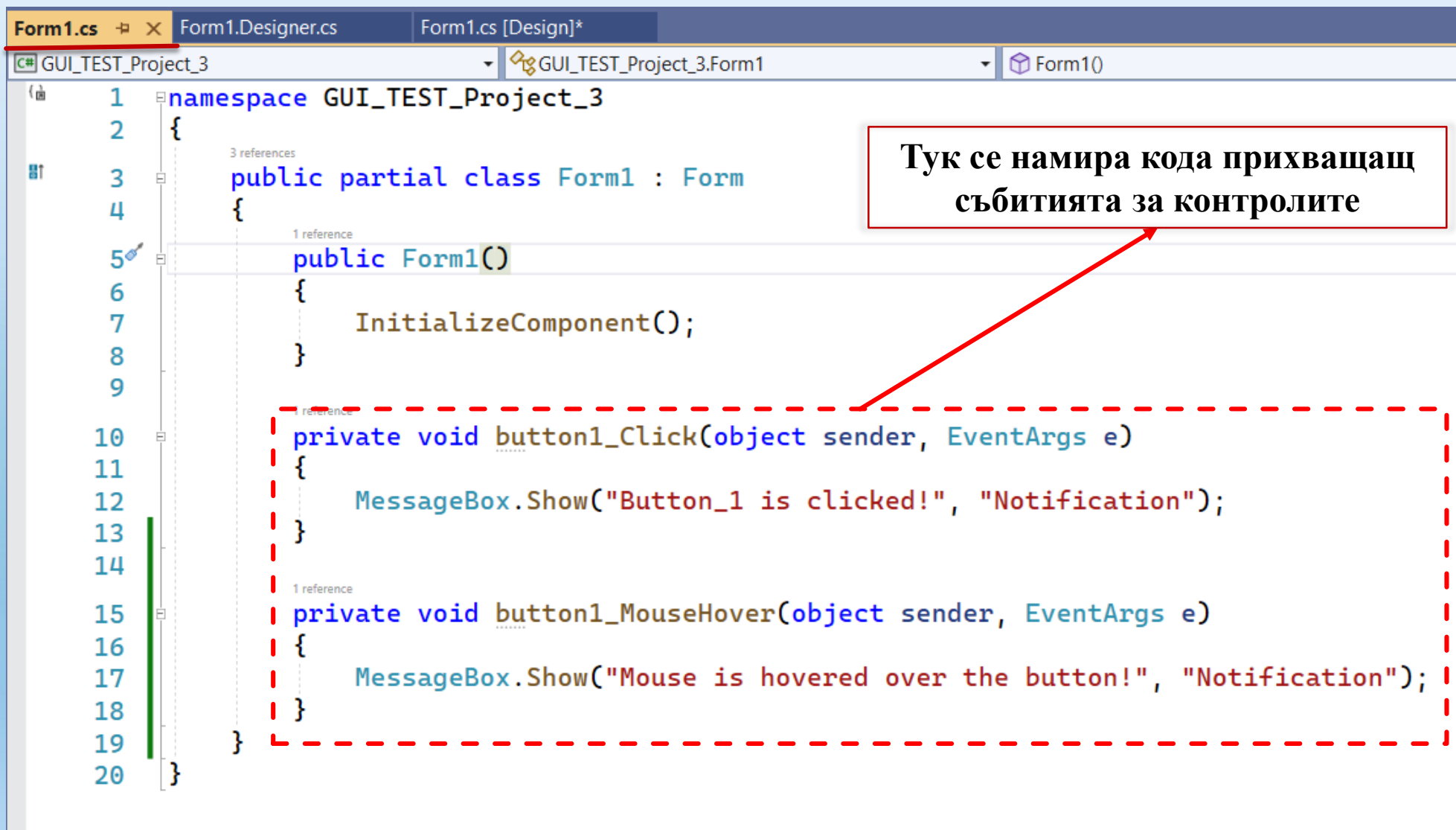
Събития се добавят като се маркира контролата, за която искаме да създадем събитие и от прозореца Properties изберем бутона за визуализиране на събития.



За да се генерира автоматично кода за прихващане на събитието е нужно да се кликне с левия бутон на мишката двукратно и бързо върху името на желаното събитие, след което VisualStudio ни препраща към файла съдържащ кода за прихващане на събитието.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ

4



```
1 namespace GUI_TEST_Project_3
2 {
3     public partial class Form1 : Form
4     {
5         public Form1()
6         {
7             InitializeComponent();
8         }
9
10        private void button1_Click(object sender, EventArgs e)
11        {
12            MessageBox.Show("Button_1 is clicked!", "Notification");
13        }
14
15        private void button1_MouseHover(object sender, EventArgs e)
16        {
17            MessageBox.Show("Mouse is hovered over the button!", "Notification");
18        }
19    }
20 }
```

Тук се намира кода прихващащ събитията за контролите

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



## *По-важните събития на класа Control:*

- ✓ **Click** – **настъпва при щракване с мишката върху контролата**. При бутон това събитие се извиква при натискане на бутона. При Форма, Click се извиква при щракване с левия бутон на мишката върху *Формата*, ако в съответната позиция няма друга контрола. Събитието не подава допълнителна информация в аргументите си.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Button_1 is clicked!", "Notification");
}
```

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ✓ **Enter, Leave** – настъпват съответно при активиране и деактивиране на дадена контрола, т.е. когато контролата получи и загуби фокуса. При Форми тези събития не се извикват.

1 reference

```
private void button1_Enter(object sender, EventArgs e)
{
    MessageBox.Show("The Button1 is entered !", "Notification");
}
```

1 reference

```
private void button1_Leave(object sender, EventArgs e)
{
    MessageBox.Show("Mouse leaved th Button 1!", "Notification");
}
```

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ✓ **KeyDown, KeyUp** – **настъпват при натискане и отпускане на произволен клавиш** (включително специалните клавиши като [F1], [Alt], [Caps Lock], [Start] и др.). Събитието подава в аргументите си инстанция на класа `KeyEventArgs`, която съдържа информация за натиснатия клавиш – име на клавиша (инстанция на изброения тип `System.Windows.Forms.Keys`) и информация за състоянието на клавишите [Shift], [Alt] и [Ctrl].
- ✓ **KeyPress** – **настъпва при натискане на неспециален клавиш или комбинация от клавиши**. Това събитие се активира само ако натиснатата клавишна комбинация се интерпретира като символ. Например натискането на клавиша [Alt] не води до получаване на символ и не задейства това събитие, докато натискането на клавиша [V] генерира някакъв символ в зависимост от текущия език. Събитието подава в аргументите си инстанция на `KeyPressEventArgs` класа, която съдържа символа, генериран в резултат от натискането на клавиша.



# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ✓ **MouseDown, MouseMove, MouseUp, MouseWheel** – настъпват при събития от мишката, извършени върху контролата – натискане на бутон, движение на показалеца на мишката или преместване на колелото. Събитията подават в аргументите си инстанция на MouseEventArgs класа, която съдържа информация за състоянието на бутоните и колелото на мишката и за координатите на показалеца (изчислени спрямо горния ляв ъгъл на контролата).
- ✓ **MouseEnter, MouseLeave, MouseHover** – настъпват при навлизане, излизане и преместване на позицията на показалеца на мишката в рамките на контролата.
- ✓ **Move** – настъпва при преместване на контролата. Преместването може да се предизвика от потребителя (например преместване на форма) или програмно (чрез промяна на свойството Location).

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ✓ **Paint** – **настъпва при пречертаване на контролата** (при обработката на съобщението WM\_PAINT). В това събитие контролата трябва да извърши пречертаването на графичния си образ. Събитието получа-ва в аргументите си инстанция на PaintEventArgs, която съдържа Graphics обекта, върху който трябва да се извърши чертането.
- ✓ **Resize** – **настъпва при промяна на размера на контролата**. Може да се предизвика както от потребителя (при преоразмеряване на форма), така и програмно (при промяна на свойството Size).
- ✓ **TextChanged** – **настъпва при промяна на свойството Text на контролата**.
- ✓ **Validating** – **използва се за валидация на данните, въведени в контролата**. Валидацията на данни ще бъде дискутирана по-късно в настоящата тема.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



## 4. Класът ScrollableControl

Класът **ScrollableControl** е наследник на класа **Control** и добавя към него функционалност за скролиране. Ето по-важните му свойства:

- ☐ **AutoScroll** – задава дали при нужда контролата ще получи автоматично скролиращи ленти.
- ☐ **HScroll, VScroll** – задават дали контролата да има хоризонтална и вертикална скролираща лента.

## 5. Класът ContainerControl

Класът **ContainerControl** осигурява функционалност за управление на фокуса. Свойството му **ActiveControl** съдържа във всеки един момент контролата, която е на фокус.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



## 6. Форми, прозорци и диалози

Формите и диалозите в **Windows Forms** са прозорци, които съдържат контроли. Те могат да бъдат различни видове: да имат или нямат рамка, да са модални или не, да са разтегливи или не, да са над всички други прозорци или не и т.н.

### 6.1 Класът *System.Windows.Forms.Form*

Класът **System.Windows.Forms.Form** е базов клас за всички форми в **Windows Forms GUI** приложенията. Той представлява графична форма - прозорец или диалогова кутия, която съдържа в себе си контроли и управлява навигацията между тях.

Повечето прозорци имат рамка и специални бутони за затваряне, преместване и други стандартни операции. Външният вид на прозорците и стандартните контроли по тяхната рамка зависят от настройките на графичната среда на операционната система. Програмистът има само частичен контрол над външния вид на прозорците.

Класът **Form** е наследник на класовете **Control**, **ScrollableControl** и **ContainerControl** и наследява от тях цялата им функционалност, всичките им свойства, събития и методи.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



## *По-важни свойства на класа Form:*

Всички прозорци и диалози в Windows Forms наследяват класа Form и придобиват от него следните свойства:

- ❑ **FormBorderStyle** – **указва типа на рамката на формата**. По-често използваните типове рамка са следните:
  - **Sizable** – **стандартна разширяема рамка**. Потребителят **може да променя размерите** на такива рамки;
  - **FixedDialog** – **диалогова рамка с фиксирани размери**. Такива рамки **не могат да се преоразмеряват** от потребителите;
  - **None** – липса на рамка. Цялото пространство на формата се използва за нейното съдържание;
  - **FixedToolWindow** – кутия с инструменти с **фиксиран размер**. Рамката **не може да се преоразмерява от потребителите** и е малко по-тъсна от стандартната. **Прозорци с такива рамки не се виждат в лентата на задачите (taskbar) на Windows Explorer и при натискане на [Alt+Tab]**.



# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **Controls** – съдържа списък с контролите, разположени във формата. От реда на контролите в този списък зависи редът, в който те се чертаят на екрана (Z-order) и редът, в който се преминава от една контрола към друга при навигация (tab order). Редът на преместване на фокуса може да се настройва и допълнително от свойствата TabStop и TabIndex.
- ❑ **Text** – заглавие на прозореца. Използва се Unicode, т.е. можем да използваме, кирилица, латиница, гръцки и други азбуки от Unicode стандарта.
- ❑ **Size** – размери на прозореца (ширина и височина). Включва цялото пространство, заемано от формата (рамката + вътрешността).
- ❑ **ClientSize** – размери на вътрешността на формата (без рамката ѝ).
- ❑ **AcceptButton** – бутон по подразбиране. Този бутон се натиска автоматично, когато потребителят натисне клавиша [Enter], независимо от това в коя контрола от формата е фокусът в този момент. Целта е да се улесни потребителя при попълването на форми с информация.



# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **ActiveControl** – съдържа контролата, която държи фокуса. При промяна на това свойство се променя текущата фокусирана контрола.
- ❑ **ControlBox** – задава дали *Формата* трябва да съдържа стандартните контроли за затваряне, минимизация и т. н.
- ❑ **Icon** – задава икона на прозореца.
- ❑ **KeyPreview** – ако се зададе true, позволява формата да обработва събитията от клавиатурата, преди да ги предаде на фокусираната контрола. Ако стойността е false, всяко събитие от клавиатурата се обработва само от контролата, която е на фокус.
- ❑ **MinimumSize, MaximumSize** – задава ограничения за размера на *Формата* – максимална и минимална ширина и височина. При опит за преоразмеряване не се позволява потребителят да задава размер, който не е в тези граници.
- ❑ **Modal** – връща дали формата е модална. Когато една форма е модална, докато тя е активна, потребителят не може да работи с други форми от същото приложение. Всеки опит за преминаване в друга форма не успява, докато потребителят не затвори модалната форма.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



Ако дадено приложение покаже едновременно няколко форми, които не са модални, потребителят ще може да преминава свободно между тях, без да ги затваря. Свойството Modal е само за четене. Модал-ността може да се задава първоначално, но не може да се променя, след като формата е вече показана.

- ☐ **Opacity** – задава прозрачност на формата (число от 0.00 до 1.00). Възможно е да не се поддържа или да работи много бавно при някои по-стари видеоадаптери.
- ☐ **MdiChildren** – в MDI режим извлича / задава подчинените форми на текущата форма. MDI (**Multiple-Document Interface**) е режим, при който дадена форма на приложението (обикновено главната форма) може да съдържа в себе си други форми, които са разположени в нейното работно пространство (като обикновени контроли).
- ☐ **MdiParent** – в MDI режим извлича / задава формата, която е собственик на текущата форма. Важи само за подчинени (child) форми.
- ☐ **TopMost** – задава дали формата стои над всички други прозорци (always on top). В такъв режим, дори ако формата не е активна, тя остава видима и стои над всички останали форми.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❑ **WindowState** – **извлича състоянието на формата**. *Формата* във всеки един момент е в някое от състоянията на изброения тип `FormWindowState` – **нормално**, **минимизирано** или **максимизирано**. По подразбиране формите са в нормално състояние – имат нормалния си размер. В максимизирано състояние формите временно променят размера си и заемат целия екран без лентата за задачи (task bar) на Windows Explorer. В минимизирано състояние формите са скрити и се виждат само в лентата за задачи (task bar).

## *По-важни методи на класа Form:*

Прозорците и диалозите в Windows Forms наследяват от класа `Form` следните базови методи:

- ❖ **Show()** – **показва формата и я прави активна (фокусира я)**. Формата се показва в немодален режим. Извикването на този метод е еквивалентно на присвояването **Visible=true**. Изпълнението на този метод приключва веднага.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



- ❖ **ShowDialog()** – показва формата в модален режим и след като тя бъде затворена, връща като резултат стойност от тип DialogResult. Тази стойност съдържа информация за **причината за затваряне на формата**. Изпълнението на метода ShowDialog() приключва едва след затваряне на формата, т.е. методът е блокиращ. По-нататък в настоящата тема ще обърнем специално внимание на извикването на модални форми и получаването на стойностите от контролите в тях.
- ❖ **Close()** – затваря формата. Когато една форма бъде затворена, тя изчезва и се **освобождават използваните от нея ресурси**. След като една форма бъде затворена, тя не може да бъде повече показвана. За временно скриване на форма трябва да се използва методът Hide(), а не Close().
- ❖ **LayoutMdi(...)** – в MDI режим този метод пренарежда дъщерните (child) форми, **съдържащи се в текущата форма**. Начинът на прена-реждане се задава от програмиста. Поддържат се няколко вида пренареждане - каскадно, хоризонтално, вертикално и др.

# I. ЙЕРАРХИЯ НА КЛАСОВЕТЕ ЗА ФОРМИ, ПРОЗОРЦИ И ДИАЛОЗИ



## *По-важни събития на класа Form*

Всички прозорци и диалози в Windows Forms поддържат съвкупност от стандартни събития, които наследяват от класа Form:

- ✓ **Activated / Deactivate** – извикват се при активиране / деактивиране на формата (когато формата получи / загуби фокуса).
- ✓ **Closing** – извиква се при опит за затваряне на формата (например, когато потребителят натисне стандартния бутон за затваряне). Реализацията може да предизвика отказване на затварянето. Събитието подава в аргументите си инстанция на класа `CancelEventArgs`, която има булево свойство `Cancel`, чрез което може да се откаже затварянето.
- ✓ **Load** – извиква се еднократно преди първото показване на формата. Може се ползва за инициализиране на състоянието на контролите.



## II. УПРАВЛЕНИЕ НА СЪБИТИЯТА



Прихващането на събитие става чрез добавянето на обработчик за него. За целта създаваме метод, който ще обработва събитието, и след това се абонираме за него. Windows Forms дизайнерът на Visual Studio .NET улеснява прихващането на събития, като генерира автоматично обработчиците при избор на събитие от страницата "Events" на прозореца "Properties".

В Windows Forms има няколко типа събития:

- **EventHandler** – извършва проста нотификация, без да подава допълнителни данни за възникналото събитие;
- **KeyEventHandler** – събития от клавиатурата. Подава се информация кой е натиснатият клавиш, както и информация за състоянието на клавишите [Ctrl], [Shift] и [Alt];
- **MouseEventHandler** – събития от мишката. Подава се информация за позицията на мишката и състоянието на нейните бутони;
- **CancelEventHandler** – събития, които могат да откажат започнатото действие. Примерно, ако прихващаме събитието Closing на дадена форма, което е от тип CancelEventHandler, и потребителят се опита да затвори формата, можем да откажем затварянето, ако данните не са запазени.



### III. ИЗПОЛЗВАНЕ НА ДИАЛОГОВИ КУТИИ



При разработката на Windows Forms приложения често пъти се налага да извеждаме диалогови кутии с някакви съобщения или с някакъв въпрос. Нека разгледаме стандартните средства за такива ситуации.

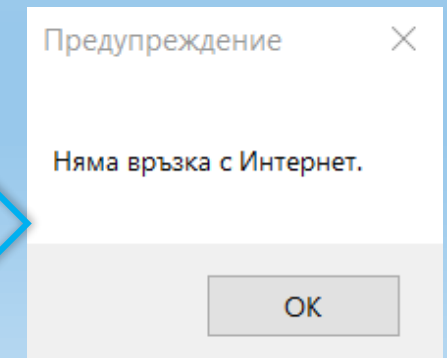
#### 1. Стандартни диалогови кутии

Класът **MessageBox** ни позволява да извеждаме стандартни диалогови кутии, съдържащи текст, бутони и икони. Стандартните диалогови кутии бива два вида:

- съобщения към потребителя;
- въпросителни диалози

Показването на диалогова кутия се извършва чрез извикване на статичния метод **Show(...)** на класа **MessageBox**. Следният код, например, ще покаже диалогова кутия със заглавие "Предупреждение" и текст "Няма връзка с интернет":

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Няма Връзка с Интернет.", "Предупреждение");
}
```

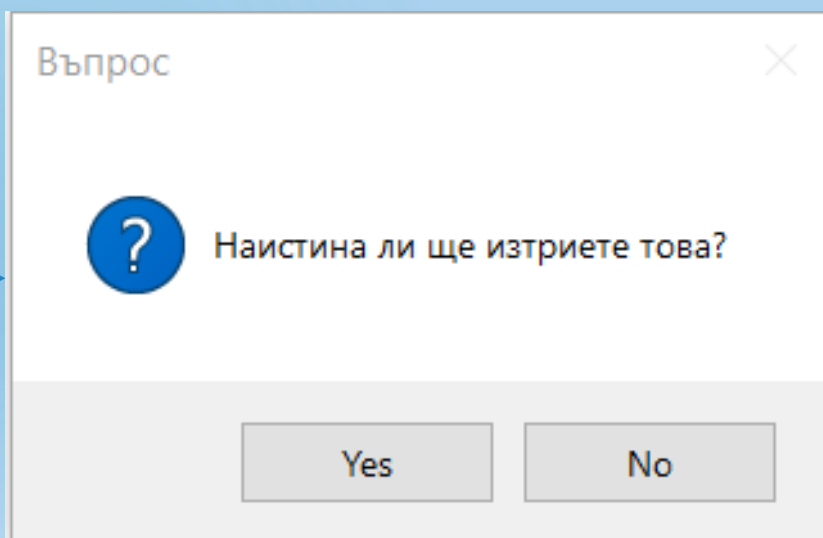


### III. ИЗПОЛЗВАНЕ НА ДИАЛОГОВИ КУТИИ



Нека разгледаме още един пример за стандартна диалогова кутия с малко повече функционалност:

```
private void button1_Click(object sender, EventArgs e)
{
    bool confirmed = MessageBox.Show("Наистина ли ще изтриете това?",
                                     "Въпрос", MessageBoxButtons.YesNo,
                                     MessageBoxIcon.Question) == DialogResult.Yes;
}
```



Ако потребителят натисне Yes, променливата `confirmed` ще има стойност `true`, в противен случай ще има стойност `false`.