

# **ВЪВЕДЕНИЕ В ПРОГРАМИРАНЕТО ПОД WINDOWS**

**ЛЕКЦИЯ 4**

# СЪДЪРЖАНИЕ

---

- ▶ Основни принципи на визуалното програмиране
- ▶ Принципи на проектиране на ГПИ
- ▶ Структура на графично Windows приложение
- ▶ Библиотеки на .NET за изграждане на ГПИ
- ▶ Програмен модел на Windows Forms
- ▶ Основни класове в Windows Forms
- ▶ Обработка на събития
- ▶ Форми, прозорци и диалози
- ▶ Класът Control

# Основни принципи на визуалното програмиране

---

- ▶ **Визуалните езици** се отнасят към програмните езици от четвърто поколение
- ▶ Разработването на този тип програмни средства се явява продължение на **две важни концепции** в развитието на програмните езици:
  - ▶ развитие на **обектно-ориентираното програмиране** (ООП), което даде възможност за разработване на универсални програмни обекти, осигуряващи многократното им използване в различни програмни приложения
  - ▶ разработване на **графичния потребителски интерфейс** за операционни системи и големи функционални възможности на свързаният с тях **стил на програмиране с управление на събития**

# Основни принципи на визуалното програмиране

---

- ▶ Избягва се тежката рутинна работа по създаване на интерфейса на програмите
- ▶ Разработва се специфичен за дадено приложение програмен код
- ▶ Основните характеристики на визуалните програмни среди се определят от спецификата на **програмиране с управление на събитията и обектно-ориентирания подход**

# Процедурно и визуално програмиране - разлики

---

- ▶ **Процедурно програмиране** – програмата се състои от набор от процедури и команди, които се изпълняват в строго определен и последователен ред, указан от програмиста
- ▶ **Програми, управлявани от събития** – програмата се състои от **набор от процедури**, всяка от които **описва действието на програмата в отговор на събитие или съобщение**, предизвикано от потребителя, ОС или друга програма

# Програмиране с управление на събития

---

- ▶ допълва процедурното програмиране със средства, които позволяват да се **отдели потребителският интерфейс от специфичната обработка на данните**
- ▶ осигурява възможност на **програмиста** да се съсредоточи върху **логиката на работа на самата програма**
- ▶ основната грижа за диалог с потребителя се предоставя от програмната среда

# Принцип за управление на събития

---

- ▶ Концепцията за програмиране с управление на събития е взета от реалния живот
- ▶ Такива процеси се срещат навсякъде около нас:
  - ▶ **сигналят при позвъняване на телефона** е събитие, което изисква определени действия. Трябва да се вдигне слушалката и да се проведе разговор
  - ▶ **показанието на часовника, отбелязващ времето** е знак, че е завършил учебният час и като отговор на това събитие студентите напускат аудиторията
  - ▶ **падането на температурата в помещението под 20°C** предизвиква включване на нагревателя на интелигентните отоплителни устройства
  - ▶ ...

# Какво е събитие?

---

- ▶ **Събитие** е такова **действие**, което протича в реално време и като отговор **предизвиква други действия**
- ▶ **Основното при програмирането с управление на събития е:**
  - ▶ **идентифицирането** на определени действия (събития) и
  - ▶ **генериране на други действия** като отговор на тези събития



# Среда за управление на събития

---

- ▶ **Не всички събития** са с еднаква важност
- ▶ Трябва **да се обработват само тези сигнали**, които имат значение за съответната програма
- ▶ Средата за управление на събития, позволява да се **идентифицира натискането на клавиш** и да се даде възможност на програмното приложение веднага да премине към **изпълнение на логически обособена група операции**, която да се свърже с определено събитие
- ▶ Програмирането с управление на събитията изисква **специфичен подход при проектиране и разработване на програмни приложения**

# Програмиране с управление на събития

---

- ▶ Програмистът най-напред обмисля **външния вид на програмата** (екранните форми за диалог с потребителя) и последователността за осъществяване на този диалог
- ▶ След това се разглеждат **всички събития**, които могат да се **случат** в процеса на работа на програмата
- ▶ Такива събития могат да бъдат:
  - ▶ натискане на бутона на мишката
  - ▶ позициониране върху определен ред от списък с елементи
  - ▶ натискане на клавиш от клавиатурата и т.н.

# Програмиране с управление на събития

---

- ▶ За всяко събитие, имащо значение за даденото приложение, се съставя отделна **процедура, обработваща събитието**
- ▶ Програмите създадени за работа в **графична среда (Windows)**, представляват **съвкупност от всички процедури, обработващи събитията**, които са възможни в процеса на работата им

# Събитийно програмиране

---

- ▶ При събитийното програмиране програмата е в аморфно (изчакващо) състояние и настъпването на определени събития определя кога и коя част от програмния код ще бъде изпълнена
- ▶ **Реакцията** на определено събитие чрез изпълнение на програмен код **не е задължителна**
- ▶ При събитийното програмиране **класовете от обекти** имат допълнителна **характеристика**:
  - ▶ **набор от събития**, на които техните екземпляри имат право да реагират

# Особености на събитията

---

Събитията възникват **асинхронно**, следователно:

- ▶ докато в отговор на дадено събитие се изпълнява програмен код за реакцията му, може **да възникне друго събитие**
- ▶ програмните участъци, отговорни за реакциите на събития по естествен начин се изпълняват паралелно
- ▶ някои **събития са взаимно свързани**, т.е. едното не може да възникне без преди това да **се е случило другото**
  - ▶ двукратното щракване с мишка се предшества от еднократно
  - ▶ разпознаването на стандартен клавиш се прешества от натискането му и др.

# Предимства на събитийното програмиране

---

Програмата, управлявана от събития е:

- ▶ по-пригодна за **диалогово изпълнение**
- ▶ по-приспособена към изискванията на съвременния ГПИ
- ▶ пригодена за **паралелно изпълнение** по естествен начин
- ▶ **по-удобна за създаване**, защото често не е необходимо писане на програмен код

# Графичен потребителски интерфейс (Graphical user interface - GUI)

---

- ▶ **Графичен потребителски интерфейс (ГПИ)**, е разновидност на **потребителски интерфейс**, в който **елементите на интерфейса** (менюта, бутони, списъци и др.), които са **предоставени на потребителя за управление**, са изпълнени във вид на графични изображения
- ▶ **ГПИ (GUI)** позволява на потребителя лесно да си **взаимодейства с програмата**, с помощта на различни визуални компоненти
- ▶ **ГПИ е стандарт** за изграждане на различни видове **съвременни софтуерни приложения**, като за целта се използва **общ набор от компоненти** на потребителския интерфейс

# Графичен потребителски интерфейс

---

- ▶ За разлика от **интерфейса с команден ред**, в ГПИ потребителят **има произволен достъп до всички видими обекти на екрана на монитора**
- ▶ Тези **обекти** се наричат **елементи на интерфейса**, като с помощта на **периферно устройство** (клавиатура, мишка и т. н.), се осъществява непосредственото им манипулиране.
- ▶ Най-често **елементите** на **графичния интерфейс** се реализират като **икони**, които **подказват тяхното предназначение и свойства**



# Елементи на графичния интерфейс

---

- ▶ **Прозорец** – визуален обект, чрез който потребителят общува с програмата – въздейства ѝ и получава резултата от нейната обработка
- ▶ **Етикет** – най-простите компоненти на графичния интерфейс. Етикетът е **текст**, изображение(**икона**) или комбинация от текст и икона
- ▶ **Икона** – изображение, което подсказва предназначението на елемент от графичния интерфейс
  - ▶ Понякога тя заменя **името на елемент**
  - ▶ Иконата служи като **бърза връзка** за стартиране на програма и за изпълнението на команда от меню
- ▶ **Бутон** – използва се за предаване на информация в програмата, управляваща прозореца, с цел изпълнение на някакво действие

# Елементи на графичния интерфейс

---

- ▶ **Меню** - списък от бутони за избор на команди. Менютата **събират на едно място** бутони, изпълняващи **логически свързани помежду си действия**.  
**Менюто** може да съдържа **команди, подменюта и специални команди** (препратки, **бърза връзка**) за стартиране на програми
- ▶ **Страница** - **средство** за съвместяване на няколко подпрозореца в един **прозорец**

# Стандартният прозорец на Windows

---

- ▶ **Основен компонент** на графичното приложение е **стандартният прозорец на Windows**
- ▶ В прозореца се разполагат **контроли на ГПИ**, които се наричат **интерфейс**
- ▶ **Контролите** са **обекти**, които дават възможност за **визуализиране на информацията на екрана** и за **взаимодействие с приложението** чрез **мишка, клавиатура и др. средства за въвеждане на данни**

- ▶ ГПИ се **управлява от събитията**
- ▶ Следователно, когато потребителят **взаимодейства с контрола** се получава **събитие**, което **се прихваща** и **се обработва от** програмата
  - ▶ **Примери** за такива **събития** са **натискане на клавиш, натискане на бутон на мишката** и др.
- ▶ **Методът**, който се изпълнява в приложението в отговор на събитие с цел **изпълнение на задача**, се нарича **обработчик на събитие**, а **процесът обработване на събитие**

# Концепцията на Windows Forms

---

- ▶ **Графичните приложения** в ОС Windows се базират на концепцията на **Windows формите**
- ▶ Това е концепция, която е взимствана от средите за бърза разработка.
- ▶ **Формата** е графичен елемент, който може да е:
- ▶ **диалогов** прозорец
- ▶ **стандартен** прозорец
- ▶ **многодокументен** прозорец

# КАКВО ПРЕДСТАВЛЯВА Windows Forms?

---

- ▶ **Windows Forms** е стандартната библиотека на .NET Framework за изграждане на **прозоречно-базиран графичен потребителски интерфейс (GUI)** за **настолни (desktop)** приложения
- ▶ **Windows Forms** дефинира **набор от класове и типове**, които позволяват **изграждане на прозорци и диалози с графични контроли** в тях, чрез които се извършва интерактивно взаимодействие с потребителя

# КАКВО Е RAD?

---

- ▶ Windows Forms е базирана на **RAD концепцията**
- ▶ **RAD е подход за разработка**, при който **приложенията се създават визуално чрез сглобяване на готови компоненти** посредством помощници и инструменти за **автоматично генериране** на голяма част от **кода**
- ▶ В резултат приложенията се разработват:
  - ▶ **много бързо**
  - ▶ **с малко ръчно писане на код**
  - ▶ **с намалени усилия от страна на програмиста**

# Компонентно-ориентираната разработка

---

- ▶ всеки **компонент** решава някаква определена **задача**, която е част от проекта
- ▶ компонентите се поставят в приложението, след което **се интегрират един с друг** чрез **настройка на техните свойства и събития**
- ▶ **свойствата** на всеки компонент определят различни **негови характеристики**
- ▶ **събитията** служат за **управление на действията**, които са предизвикани от тях



# Windows Forms

---

- ▶ типична **компонентно-ориентирана библиотека** за създаване на GUI
- ▶ предоставя възможност с **малко писане на програмен код** да се създава **гъвкав графичен потребителски интерфейс**
- ▶ позволява създаването на **форми и други елементи от графичния интерфейс** на приложенията да се извършва **визуално и интуитивно** чрез подходящи редактори, като например **Windows Forms Designer във Visual Studio .NET**

# КОНТРОЛИТЕ В Windows Forms

---

**Windows Forms** съдържа богат набор **от стандартни контроли**:

- ▶ **форми, диалози, бутони, контроли за избор, текстови полета, менюта, ленти с инструменти, статус ленти и много други**
- ▶ **предоставя възможност за създаване на собствени контроли**, които да се използват в приложенията
- ▶ **в интернет могат да се намерят **безплатно** или срещу **лицензна такса** голям брой библиотеки от контроли**, които решават **често срещани проблеми** и спестяват време на разработчика

# Windows Forms и работа с данни

---

- ▶ **Windows Forms** предоставя много **контроли за визуализация и редактиране на данни** – текстови, списъчни и таблични
- ▶ За **спестяване на време** на разработчика е въведена концепцията "**свързване на данни**" (**data binding**), която позволява **автоматично свързване на данните** с контролите за тяхната визуализация
- ▶ Всички контроли са съобразени с **Unicode стандарта** и позволяват използване на много **езици и азбуки** (латиница, кирилица, гръцки, арабски и др.) **без допълнителни настройки** на Windows или на приложението

# Наследяване на форми и контроли

---

- ▶ **Windows Forms** е проектирана така, че да позволява лесно наследяване и разширяване **на форми и контроли**
- ▶ **ActiveX контролите** представляват графични компоненти. Те **имат свойства**, чрез които се задават различни **характеристики** и **събития, управляващи поведението им**
- ▶ **ActiveX контролите** много приличат на **Windows Forms контролите** от .NET Framework, но за разлика от тях се **реализират с неуправляван код** и преди използване трябва **да се регистрират чрез добавяне в регистрите на Windows** (Windows Registry)

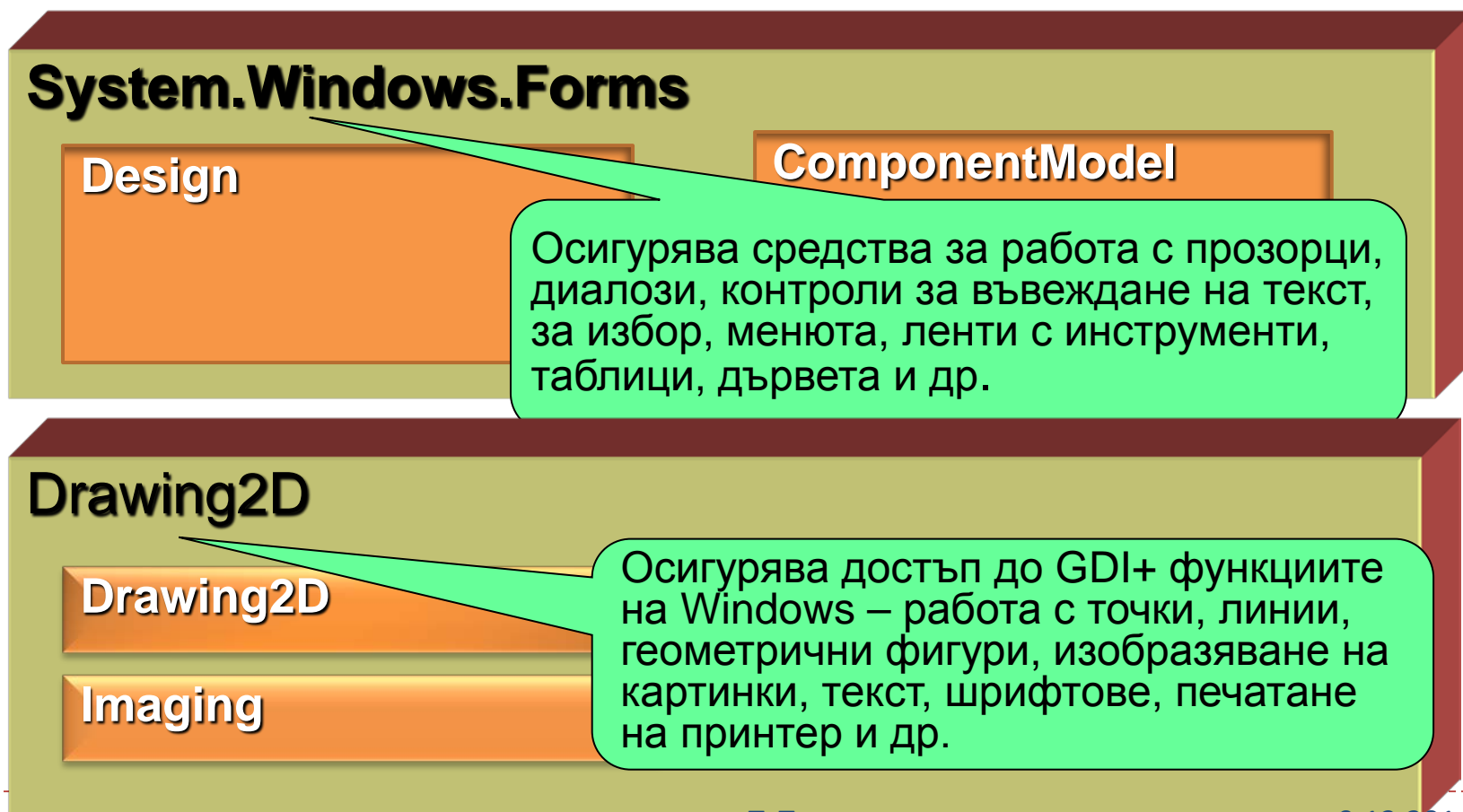
# Силна поддръжка на графика (GDI+)

---

- ▶ Библиотеката **Windows Forms** широко използва средствата на Windows платформата **за чертане и работа с графични обекти (GDI+)**
- ▶ **Windows Forms** позволява тези средства да се използват за **създаване на собствени изображения** върху различни повърхности – **в прозорец, върху принтер, плотер и др.**
- ▶ Дава се **достъп** до всички по-важни **примитиви за чертане** – **текст, графични изображения, геометрични фигури** (точки, линии, правоъгълници, елипси) и т. н.

# Библиотеките на .NET за изграждане на ГПИ (GUI)

Средствата за **изграждане на ГПИ** са дефинирани в пространствата от имена **System.Drawing** и **System.Windows.Forms**



# Пространства от класове

---

- ▶ Класовете и типовете от пространството **System.Windows.Forms** осигуряват средства за работа с прозорци, диалози, контроли за въвеждане на текст, контроли за избор, менюта, ленти с инструменти, таблици, дървета и др.
- ▶ Пространството **System.Windows.Forms.Design** съдържа **класове**, които поддържат **конфигурирането на компонентите** и **дефинират поведението на Windows Forms контролите по време на дизайн**

## Пространството **System.Drawing**

- ▶ Класовете и типовете от пространството **System.Drawing** и неговите подпространства осигуряват **достъп до GDI+ функциите на Windows**:
  - ▶ **работа с повърхности, точки, линии, четки, моливи, геометрични фигури, картинки, текст и шрифтове и др.**

# Програмни компоненти

---

- ▶ В софтуерното инженерство **компонентите са преизползваеми (reusable)** програмни единици (класове), които **решават специфична задача**
- ▶ Всеки компонент има ясно дефиниран интерфейс, който описва **неговите свойства, методи и събития**
- ▶ **Компонентен модел** - дефинира **стандартите за разработка и използване** на програмните компоненти и техния жизнен цикъл



# Компонентният модел на .NET Framework

- ▶ Компонентният модел на .NET Framework дефинира **програмния модел** (система от правила) **за създаване и използване на .NET компоненти**
- ▶ този програмен модел се реализира чрез определени **класове и интерфейси**, които поддържат описанието на компонентите
- ▶ позволява дефиниране на поведението на компонентите по
  - ▶ **време на дизайн (design-time behavior)** и по
  - ▶ **време на работа (runtime behavior)**
- ▶ **Компоненти и контейнери** - в .NET Framework са дефинирани **два вида** преизползваеми обекти:
  - ▶ **компонентите** са функционални единици, които **решават някаква задача**
  - ▶ **контейнерите** са обекти, които съдържат **списък от компоненти**

# Преизползваемост на компонентите

---

- ▶ .NET компонентите могат **директно да се преизползват** във **всички .NET езици** за програмиране

## Пространството System.ComponentModel

- ▶ Компоненти се използват не само в **Windows Forms**, а **навсякъде в .NET Framework**
- ▶ **Основната функционалност** на компонентния модел на .NET се намира в пространството **System.ComponentModel**
- ▶ В него са дефинирани основните интерфейси **IComponent** и **IContainer** и техните **имплементации Component** и **Container**

# Windows Forms и компонентният модел на .NET

---

- ▶ Компонентният модел на .NET дефинира **компоненти** и **контейнери**

**Контролите** в Windows Forms са всички **компоненти**, които са **видими** за потребителя (имат **графично изображение**)

Те биват **два вида**:

- ▶ **контроли** (бутони, текстови полета, етикети, списъчни контроли и т.н.)
- ▶ **контейнер контроли** (форми, диалози, панели и т.н.)

**Контейнерите** са предназначени да **съдържат в себе си други контроли** (включително и други контейнер контроли)

# Програмен модел на Windows Forms

---

Програмният модел на **Windows Forms** дефинира:

- ▶ **класовете за работа с форми**
- ▶ **диалози и контроли**
- ▶ **събитията на контролите**
- ▶ **жизнения цикъл на приложенията**
- ▶ **модела на пречертаване на контролите**
- ▶ **модела на получаване и обработка на събитията**
- ▶ **модела на управление на фокуса**

# Програмен модел на Windows Forms

## Форми

- ▶ **Windows Forms** предлага **стандартни класове** за работа с **форми** (това са **прозорците** и **диалозите** в GUI приложенията). **Формите** могат да бъдат **модални** и **немодални** (по една или по много активни едновременно).

**Формите са контейнер-контроли** и могат да **съдържат други контроли**, например **етикети**, **текстови полета**, **бутони** и т.н.

**Базов клас** за всички форми е класът **System.Windows.Forms.Form**.

## Контроли

- ▶ **Контролите в Windows Forms** са **текстови полета**, **етикети**, **бутони**, **списъци**, **дървета**, **таблици**, **менюта**, **ленти с инструменти**, **статус ленти** и много други.

**Windows Forms** дефинира базови класове за **контролите и класове-наследници** за всяка контрола

**Базов клас** за всички контроли е класът **System.Windows.Forms.Control**.

Пример за контрола е например бутонът (класът **System.Windows.Forms.Button**)

# Програмен модел на Windows Forms

---

## Събития

- ▶ Всички **контроли** от **Windows Forms** дефинират **събития**, които програмистът може **да прихваща**

Например:

- ▶ контролата **Button** дефинира събитието **Click**, което се активира при натискане на бутона
- ▶ контролата **TextBox** дефинира събитието **TextChanged** (текстът е променен)
- ▶ контролата **ListBox**
- ▶ .....
- ▶ **Събитията в Windows Forms** управляват взаимодействието между програмата и контролите и между самите контроли

# Основни класове в Windows Forms

---

- ▶ **System.ComponentModel.Component** – представлява **.NET КОМПОНЕНТ**
  - ▶ Използва се за реализация на неграфични компоненти.  
Например **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
- ▶ **System.Windows.Forms.ScrollableControl** – представлява **контрола**, която **поддържа скролиране на съдържанието си**
  - ▶ Може да съдържа в себе си други контроли

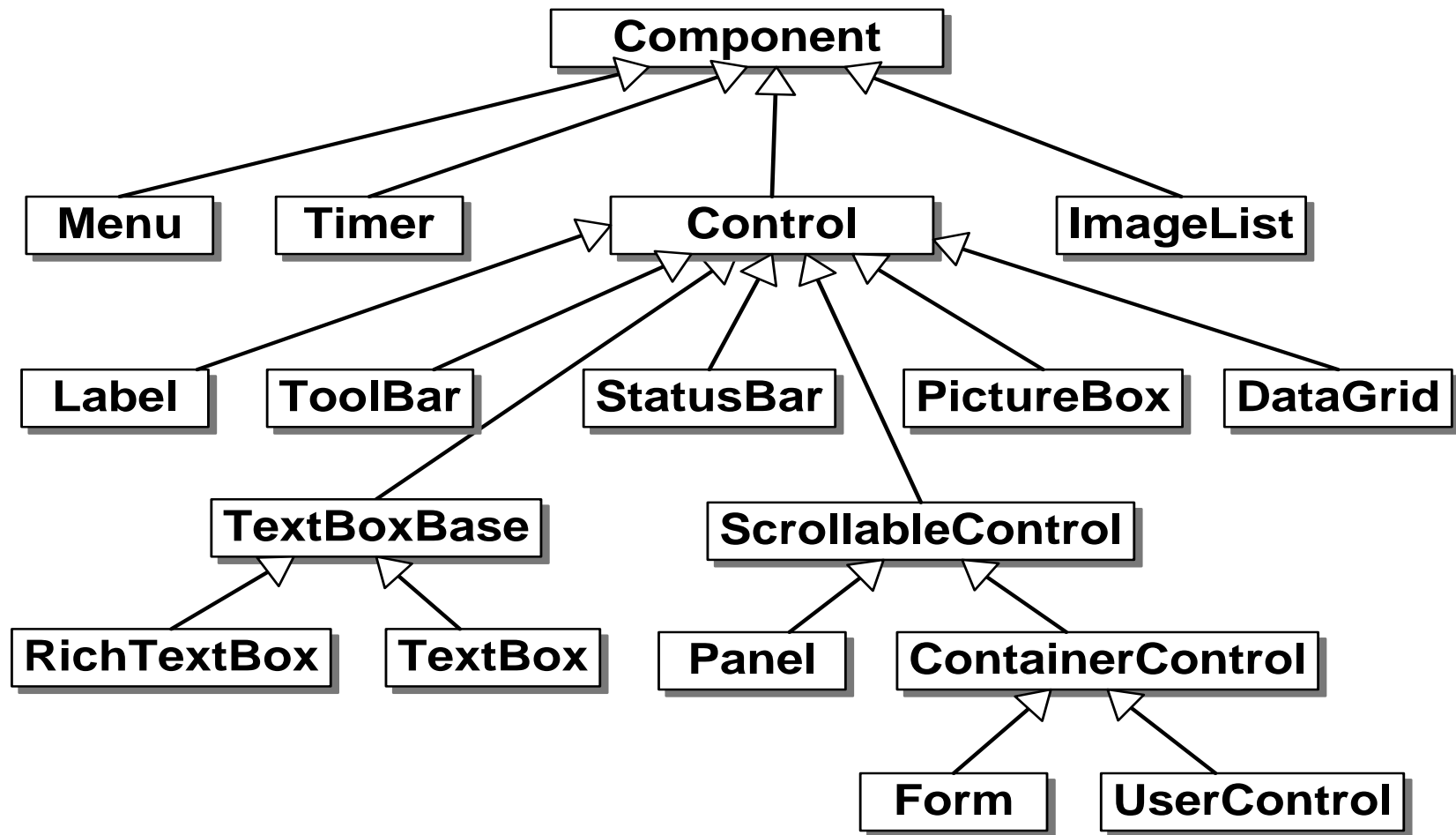
# Основни класове в Windows Forms

---

- ▶ **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**



# Йерархия на класовете



# Свойства на класа Control

- ▶ Класът **System.Windows.Forms.Control** заема **централна роля** в **библиотеката Windows Forms**. Той е **базов клас**, основа за всички графични контроли, и определя **единна рамка за контролите** – **програмен модел**, по който да се **разработват и изпълняват**
- ▶ В него са дефинирани **общите за всички контроли свойства и събития**

Свойство	Описание
Anchor, Dock	Задават по какъв начин <b>контролата се "закотвя"</b> за контейнера си. Тези свойства са много полезни, ако искаме <b>да управляваме размерите и позицията на контролата</b> при промяна на размерите на контейнера, в който е поставена.
Bounds	Задава <b>размера (ширина и височина) и позицията на горния ляв ъгъл</b> на контролата в рамките на нейния контейнер. Ако контролата е форма, позицията се задава спрямо горния ляв ъгъл на екрана. Ако контролата е елемент от форма (например бутон), позицията се отчита спрямо горния ляв ъгъл на формата (или контейнер-контролата), в която е оставена. Размерът включва цялото графично пространство на контролата. Например, ако контролата е форма, се включва и нейната рамка.

Свойство	Описание
Name	<p>Всеки компонент <b>има уникално име</b>, което се показва най-горе в прозореца. <b>Средата създава служебно име от името на класа</b>, като променя първата буква от главна в малка и добавя уникален номер (например <b>button1</b>, <b>button2</b> ... компоненти от <b>класа Button</b>)</p> <p>Програмистът може да промени името, така че да подсказва предназначението на компонента.</p>
BackColor	<p>Задава <b>цвета на фона на формата</b>. Дясното поле на свойството е комбинирана текстова кутия, в списъка на която са поставени възможните цветове. Цветовете са инстанции на структурата <b>System.Drawing.Color</b>, която дефинира множество стандартни цветове и позволява потребителски дефинирани цветове.</p>
ForeColor	<p>Задава <b>цвета, с който по премълчаване се изписват текстовете</b> на всички, включени във формата компоненти. За всяка компонента може да се зададе различен цвят.</p>
ControlBox	<p><b>Активира/деактивира групата от три бутона</b>, които управляват показването на прозореца.</p>
Enabled	<p>Указва <b>състояние на достъпност</b>.</p> <p>Промяната на това свойство от <b>true</b> във <b>false</b> оставя компонентата видима, но я прави недостъпна..</p>

Свойства	Описание
Location	Задава <b>позицията на елемента на управление (X и Y)</b> , относително спрямо горният ляв ъгъл на компонента в съдържащият елемент – форма или друг контейнер
Parent	Задава <b>контейнер-контролата, в която се намира текущата контрола</b> . Може и да няма такава (стойност null). Формите най-често <b>имат стойност null</b> за свойството Parent
Size	Задава <b>ширина (Wight) и височина (height) на компонента</b> в пиксели. Пред имената на съставните свойства е поставен триъгълният знак, с който се показват или скриват подсвойствата
TabIndex	Определя <b>реда при навигация</b> с [Tab] и [Shift+Tab]
TabStop	Задава дали <b>контролата трябва да се фокусира при навигация с [Tab] и [Shift+Tab]</b> . Ако се зададе <b>TabStop=false</b> , фокусът не спира в контролата при преминаване към следващата контрола (контролата се прескача)
Text	Задава <b>текст, свързан с контролата</b> . При <b>етикет</b> това е <b>текстът</b> , изобразен в етикета. При <b>бутон</b> това е <b>текстът, изобразен в бутона</b> . При <b>текстово поле</b> това е <b>текстът, въведен в полето</b> . При <b>форма</b> това е <b>заглавието на формата</b> . Текстът е в Unicode и това позволява да се използват свободно букви и знаци на <b>латиница, кирилица, гръцки, арабски и други азбуки</b> , стига избраният шрифт да съдържа съответните знаци.
Visible	Когато стойността е <b>true</b> , <b>компонентата е видима</b> , т.е. изобразява се в прозореца, а ако е) <b>false</b> – <b>не се изобразява</b> (може да се скриват или показват компоненти

# Методи на класа System.Windows.Forms.Control

---

- ▶ По **премълчаване** средата дава на екранната форма **име Form1**.
- ▶ **Мястото на основния прозорец** на програмата се задава в свойството **StartPosition**. По премълчаване средата дава на това свойство стойността **WindowsDefaultLocation**, т.е. там където ОС прецени за най-подходящо

Метод	Описание
Focus()	Фокусира контролата (ако е възможно)
Hide()	<b>Превръща</b> състояние на <b>видимост</b> в „невидим“.
Show()	<b>Превръща</b> състояние <b>невидим</b> във „видим“

# Събития на класа Control

Събитие	Описание
Click	<b>Кликуване с ляв бутон на</b> мишката върху елемента на управление. При <b>бутон</b> това събитие се извиква при натискане на бутона. При форма <b>Click</b> 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
DoubleClick	<b>Двойно щракване</b> с ляв бутон на мишката
Enter, Leave	Настъпват съответно при <b>активиране и деактивиране</b> на дадена контрола, т.е. <b>когато контролата получи и загуби фокуса</b> . При форми тези събития не се извикват
KeyDown, KeyUp	Настъпват при <b>натискане и отпускане на произволен клавиш</b> (включително специалните клавиши като [F1], [Alt], [Caps Lock], [Start] и др.)
KeyPress	<b>Натиснат клавиш от клавиатурата</b> , докато <b>контролата е на фокус</b> . Това събитие се активира само, ако натиснатата клавишна комбинация се интерпретира като символ
Move	Настъпва при <b>преместване на контролата</b> . Преместването може да се <b>предизвика от потребителя</b> (например преместване на форма) или <b>програмно</b> (чрез промяна на свойството <b>Location</b> )
Resize	<b>Промяна размера на контролата</b> . Може да се предизвика, както от потребителя (при преоразмеряване на форма), така и програмно (при промяна на свойството <b>Size</b> )
TextChanged	Настъпва при <b>промяна на свойството Text</b> на контролата
Validating	Използва се за <b>валидация на данните</b> , въведени в контролата

# Форми, прозорци и диалози

## Класът `System.Windows.Forms.Form`

---

- ▶ **Формите и диалозите** в `Windows Forms` са **прозорци**, които съдържат контроли
- ▶ 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**
- ▶ 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

# По-важни свойства на класа Form

СВОЙСТВО	Описание
FormBorderStyle	Указва <b>типа на рамката на формата</b> . По-често използваните типове рамка са следните: <b>Sizable</b> – стандартна разширяема рамка. Потребителят може да променя размерите на такива рамки. <b>FixedDialog</b> – диалогова рамка с фиксирани размери. Такива рамки не могат да се преоразмеряват от потребителите. <b>None</b> – липса на рамка. Цялото пространство на формата се използва за нейното съдържание. <b>FixedToolWindow</b> – кутия с инструменти с фиксиран размер. Рамката не може да се преоразмерява от потребителите и е малко по-тясна от стандартната. Прозорци с такива рамки не се виждат в лентата на задачите (taskbar) на Windows Explorer и при натискане на [Alt+Tab].
Controls	<b>Съдържа списък с контролите</b> , 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 <b>TabStop</b> и <b>TabIndex</b>
Text	<b>Заглавие на прозореца</b> . Използва се Unicode, т.е. можем да използваме, кирилица, латиница, гръцки и други азбуки от Unicode стандарта.
Size	Задава <b>размери на прозореца</b> (ширина и височина). Включва цялото пространство, заемано от формата (рамката + вътрешността).



# По-важни свойства на класа Form

свойство	Описание
ClientSize	Задава <b>размери на вътрешността на формата</b> (без рамката ѝ)
AcceptButton	Бутон по подразбиране. Този бутон <b>се натиска автоматично</b> , когато потребителят натисне клавиша <b>[Enter]</b> , 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
ActiveControl	Съдържа <b>контролата, която държи фокуса</b> . При промяна на това свойство се променя текущата фокусирана контрола
ControlBox	Задава дали <b>формата трябва да съдържа стандартните контроли</b> за затваряне, минимизация и т. н.
Icon	Задава <b>икона на прозореца</b>
KeyPreview	Ако се зададе <b>true</b> , позволява формата да обработва събитията от клавиатурата, преди да ги предаде на фокусираната контрола. Ако стойността е <b>false</b> , всяко събитие от клавиатурата се обработва само от контролата, която е на фокус
MinimumSize, MaximumSize	Задава <b>ограничения за размера на формата</b> – 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 <b>формата е модална</b> . Когато една форма е модална, докато тя е активна, <b>потребителят не може да работи с други форми</b> от същото приложение. Свойството <b>Modal</b> е <b>само за четене</b> . Модалността може да се задава първоначално, но не може да се променя, след като формата е вече показана.
WindowState	Извлича <b>състоянието на формата</b> . Формата във всеки един момент е в някое от състоянията на изброения тип <b>FormWindowState</b> – нормално, минимизирано или максимизирано. По подразбиране формите са в нормално състояние – имат нормалния си размер.

# По-важни методи на класа Form

Метод	Описание
Close()	<b>Затваря формата.</b> Когато една форма бъде затворена, тя <b>изчезва и се освобождават използваните от нея ресурси</b> . След като една форма бъде затворена, тя не може да бъде повече показвана. За <b>временно скриване</b> на форма трябва да се използва методът <b>Hide()</b> , а не Close().
Show()	<b>Показва формата</b> и я прави <b>активна</b> (фокусира я). Формата се показва в немодален режим. Извикването на този метод е еквивалентно на присвояването <b>Visible=true</b> . Изпълнението на този метод приключва веднага.
ShowDialog()	Показва <b>формата в модален режим</b> и след като тя бъде затворена, <b>връща като резултат</b> стойност от тип <b>DialogResult</b> . 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.
LayoutMdi(...)	В MDI режим този метод <b>пренарежда дъщерните (child) форми</b> , съдържащи се в текущата форма. Начинът на пренареждане се <b>задава от програмиста</b> . Поддържат се няколко вида пренареждане - каскадно, хоризонтално, вертикално и др.

# По-важни събития на класа Form

Събитие	Описание
Activated/ Deactivate	Извикват се при <b>активиране/деактивиране на формата</b> (когато формата получи/загуби фокуса)
Closing	Извиква се при <b>опит за затваряне на формата</b> (например, когато потребителят натисне стандартния бутон за затваряне). Реализацията може да предизвика отказване на затварянето. Събитието подава в аргументите си инстанция на класа <b>CancelEventArgs</b> , която има булево свойство <b>Cancel</b> , чрез което може да се откаже затварянето.
Load	<b>Извиква се еднократно</b> преди първото показване на формата. Може да се използва за <b>инициализиране на състоянието на контролите</b>

# Управление на събитията

---

- ▶ **Windows Forms** дизайнерът на Visual Studio .NET генерира **автоматично обработчиците на събития**
- ▶ **Типове събития** в Windows Forms:
  - ▶ **EventHandler** – проста нотификация
    - ▶ без допълнителни данни
  - ▶ **KeyEventHandler** – събития от клавиатурата
    - ▶ подава се кой е натиснатият клавиш и състоянието на [Ctrl], [Shift] и [Alt]
  - ▶ **MouseEventHandler** – събития от мишката
    - ▶ подава се позицията на мишката и състоянието на бутоните ѝ
  - ▶ **CancelEventHandler** – събития, които могат да откажат започнало действие

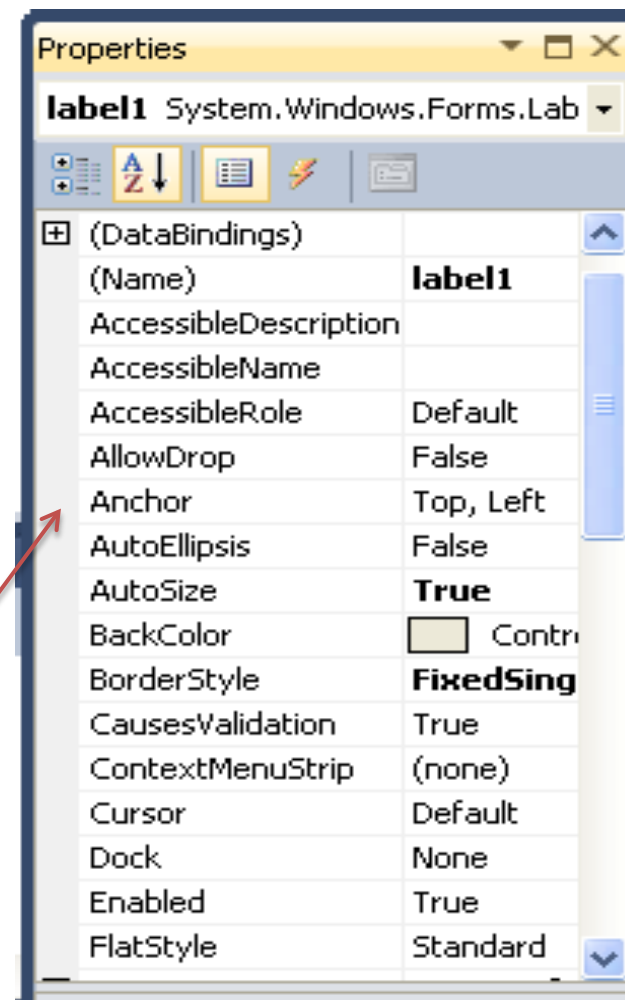
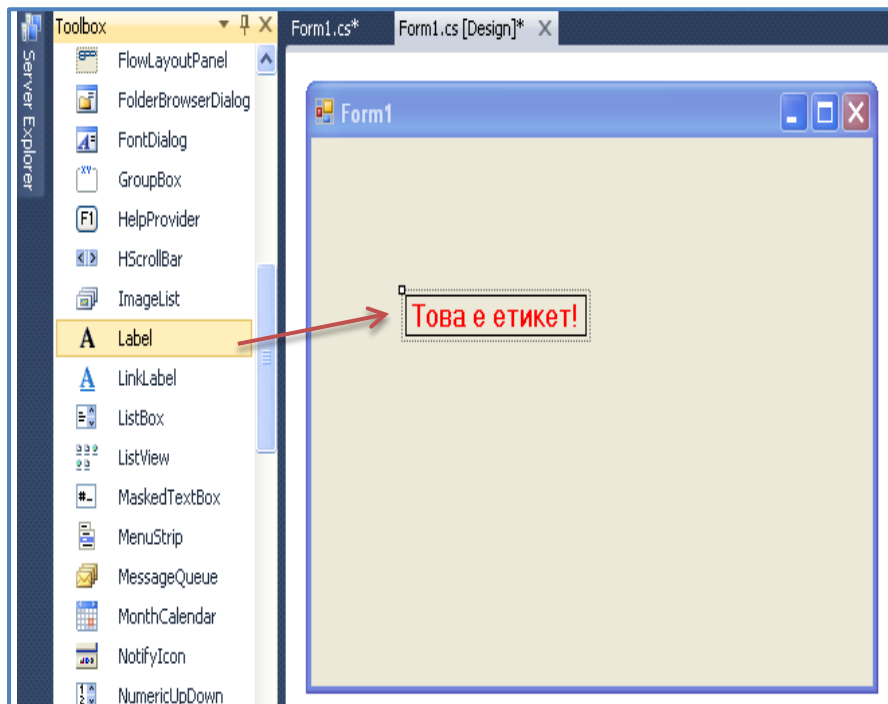
# Основни контроли в Windows Forms

## Етикет (Label) - System.Windows.Forms.Label

- ▶ Основното предназначение на компонентите от класа **Label** е да се **поставят надписи в основния прозорец** на програмата и **останалите контейнери**.
- ▶ В етикета се поставят **различни заглавия на прозорците** и **поясняващи надписи** за предназначението на останалите компоненти.
- ▶ В етикета могат да се извеждат и стойности, които трябва само да се покажат в прозореца, но **не бива да се променят от потребителя** – например, резултати от извършените от програмата пресмятания
- ▶ **Текстът**, който **ще се изпише в етикета** се задава в свойството **Text**
- ▶ При поставяне на етикет в екранната форма **Text = label1**
- ▶ Друго важно **свойство** е **Font** – съставно свойство, в което се задават параметрите на използвания шрифт, с много подсвойства. Най-важните от тях са **Name** и **Size**, задаващи вида и размера на шрифта, както и определящите стила - **Bold**, **Italic** и **Underline**

# Създаване на етикет (Label)

► `Label1.Text = "Това е етикет!";`



► Настройка на свойства

# Основни контроли в Windows Forms

## Текстова кутия (TextBox)

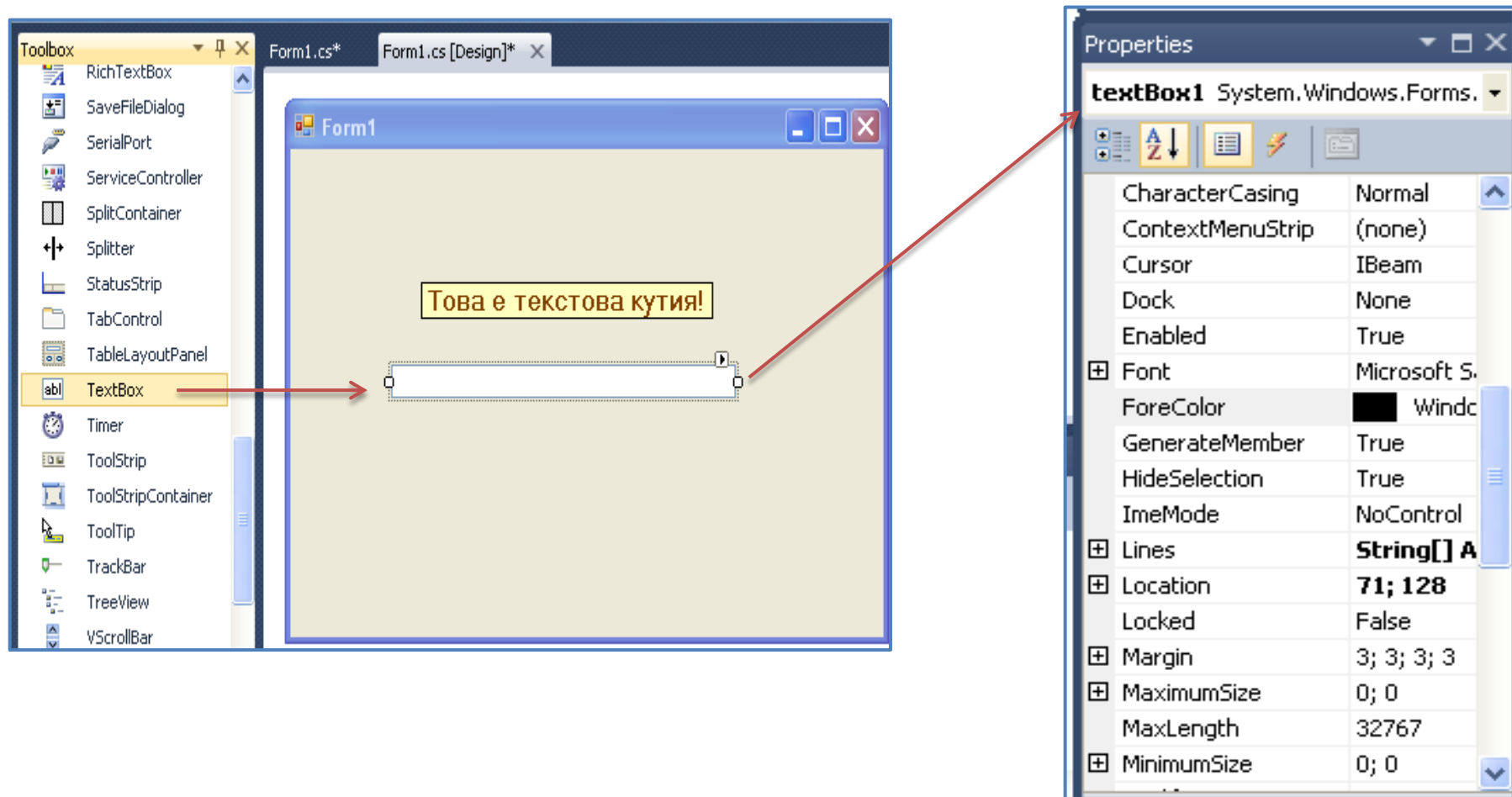
---

- ▶ Компонентите на класа **TextBox** (текстова кутия) са предназначени **за въвеждане на данни от клавиатурата по време на изпълнение на програмата**
- ▶ Това е и **основното различие между текстовата кутия и етикета**

По-важните **свойства** на **TextBox** са:

- ▶ **Multiline** – задава дали контролата представлява само един ред или допуска въвеждането на няколко реда текст.
- ▶ **Text** – съдържа въведения в контролата текст. Когато свойството **Multiline** е **true**, за достъп до въведения текст може да се използва и свойството **Lines**.
- ▶ **Lines** – масив от символни низове, съдържащ въведения текст. Всеки елемент от масива съдържа един от редовете на текста.
- ▶ Компонентите, в които потребителят може да въвежда данни, се наричат **активни**
- ▶ Важно **събитие** за текстовата кутия е **TextChanged** (текстът е променен)
- ▶ Когато това събитие се случи, програмата би трябвало да го обработи, като **съхрани** въведеното в кутията в **съответна променлива**

# Създаване на текстова кутия (TextBox)





# Основни контроли в Windows Forms

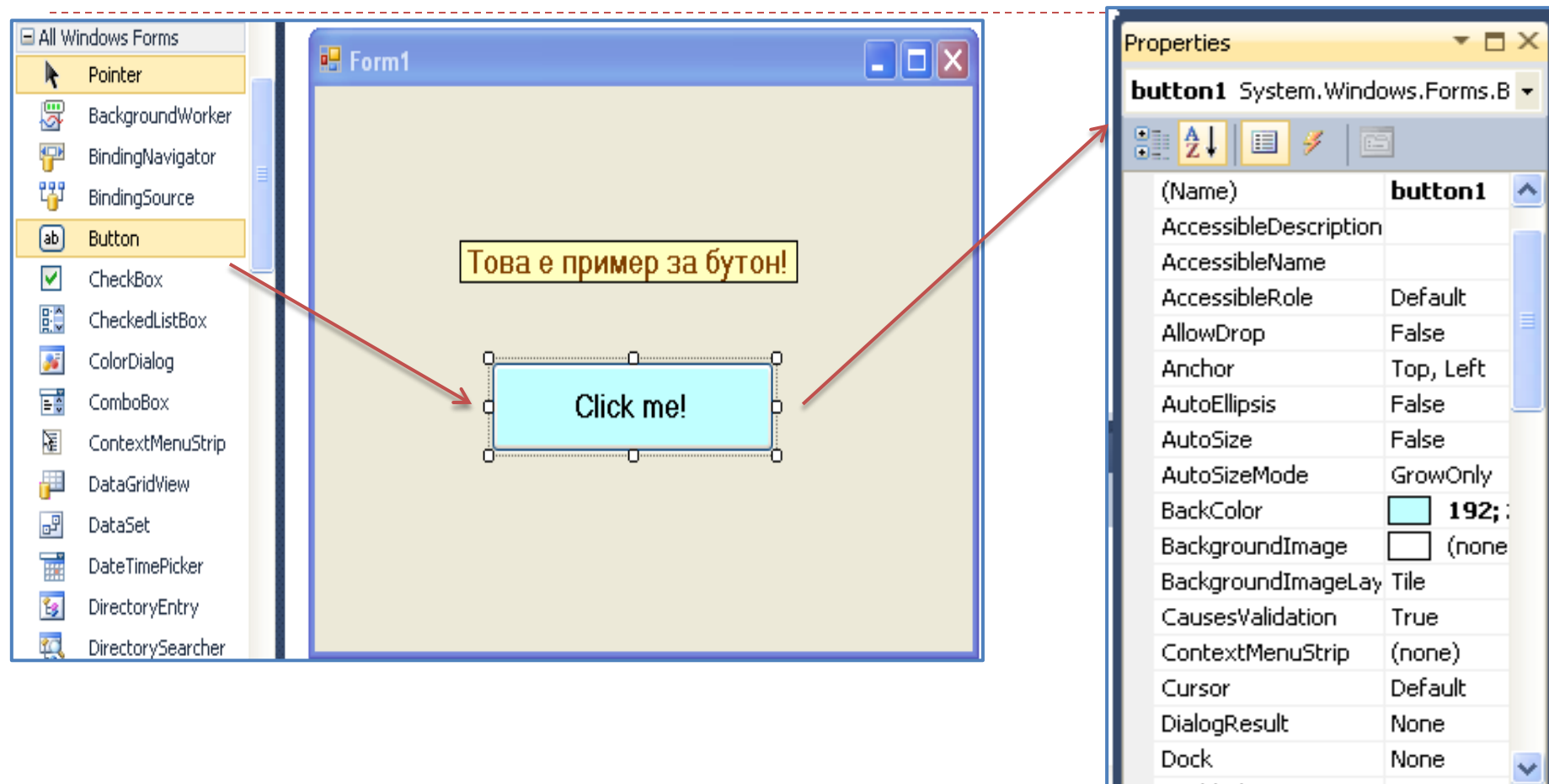
---

## ▶ Бутон (Button)

- ▶ Компонентите на класа **Button** са предназначени за подаване на команди от страна на потребителя към изпълняваната програма
- ▶ Основно **свойство** на командния бутон е **надписът му**, което е стойност на **свойството Text**
- ▶ Добре е свойството **Text** да подсказва **командата**, която ще се стартира, когато потребителят щракне върху него с мишката
- ▶ **Събитието Click** - генерира се **при натискане на бутона**
- ▶ **Средата автоматично генерира тяло на метода**, с който това събитие да бъде обработено, когато щракнем с бутона.

В тази функция програмистът изписва програмния код, който изпълнява свързаната с бутона команда

# Създаване на бутон (Button)



The image illustrates the process of creating a Button control in a Windows Forms application. It shows the Visual Studio IDE with the 'All Windows Forms' toolbox on the left, the 'Form1' design view in the center, and the 'Properties' window on the right.

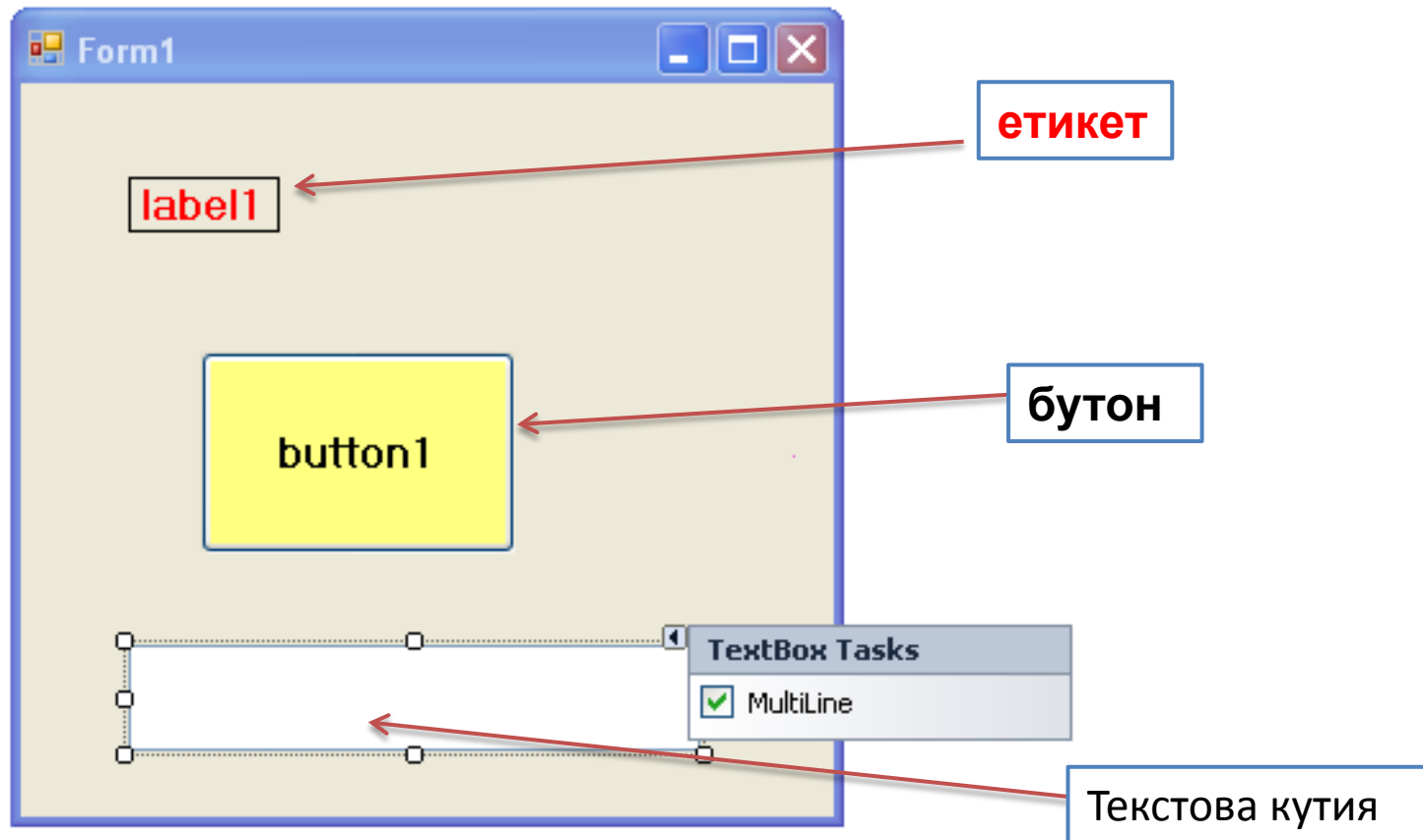
**Toolbox:** The 'All Windows Forms' toolbox is visible on the left. The 'Button' control is selected, indicated by a red arrow pointing to the 'Click me!' button in the design view.

**Design View:** The 'Form1' design view shows a light blue button with the text 'Click me!'. A yellow callout box above the button contains the text 'Това е пример за бутон!' (This is an example of a button!).

**Properties Window:** The 'Properties' window on the right shows the properties for the selected 'button1' control. The properties are listed in a table:

(Name)	button1
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoEllipsis	False
AutoSize	False
AutoSizeMode	GrowOnly
BackColor	192; ;
BackgroundImage	(none)
BackgroundImageLay	Tile
CausesValidation	True
ContextMenuStrip	(none)
Cursor	Default
DialogResult	None
Dock	None

# Примерна форма с етикет, текстова кутия и бутон



# Windows Forms редакторът на VS.NET

---

## Windows Forms Designer

**Windows Forms** редакторът на **VS.NET** ни дава възможност лесно да се извършват следните операции:

- ▶ **създаване** на форми
- ▶ **добавяне на контроли** във формите
- ▶ **добавяне на неграфични компоненти** във формите
- ▶ **настройка на свойствата** на форми, компоненти и контроли
- ▶ **добавяне на събития** за форми, компоненти и контроли

Полетата показващи някакви данни, могат да бъдат свързвани с различни източници на данни, като **бази данни** или **заявки**

- ▶ **Потребителският интерфейс** задължително се **свързва с програмен код** за създаването на приложение

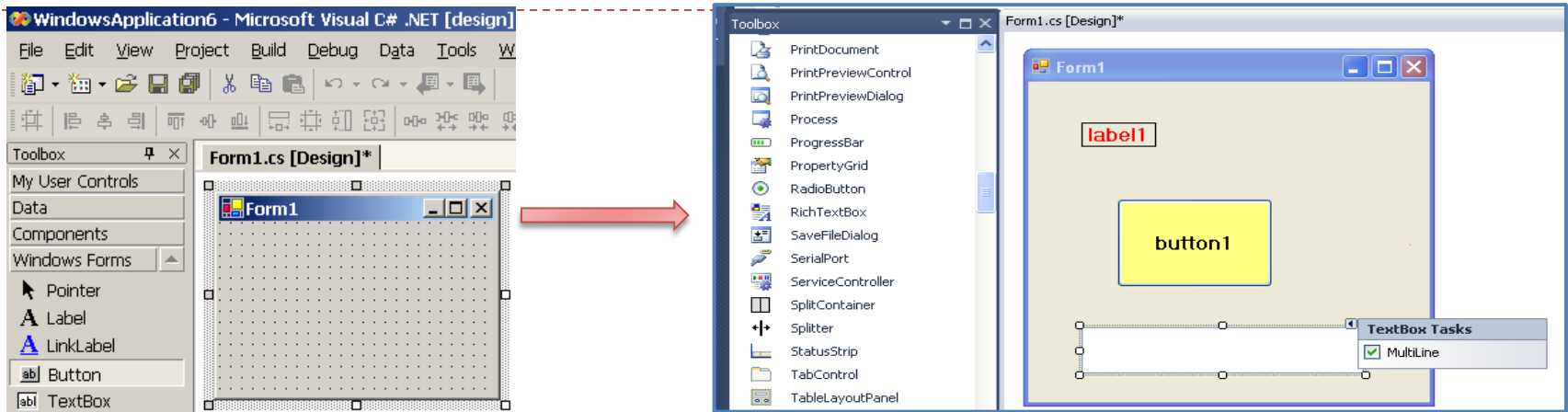
# WPF Designer

---

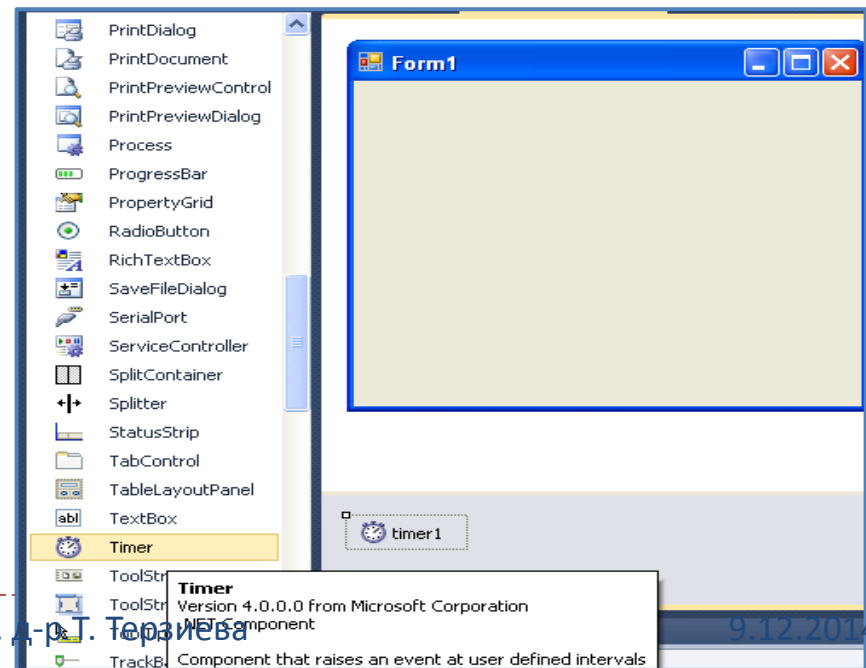
- ▶ **WPF дизайнерът**, наричан още **Cider**, базиран на **Windows Presentation Foundation**, за първи път се появява във Visual Studio 2008
- ▶ Също както Windows Forms, WPF дизайнерът се използва **за създаване на потребителски интерфейс**
- ▶ Дизайнерът поддържа **всички WPF функционалности**, включително възможността за **свързване на данни (data-binding)** и **автоматичното управление на изгледа**
- ▶ **WPF дизайнера** генерира **XAML код** за потребителския интерфейс
- ▶ Генерираният файл е съвместим с Microsoft Expression Design, продукт ориентиран към дизайнерите.
- ▶ XAML кода се свързва с кода на потребителя чрез модела code-behind

# Windows Forms и VS.NET

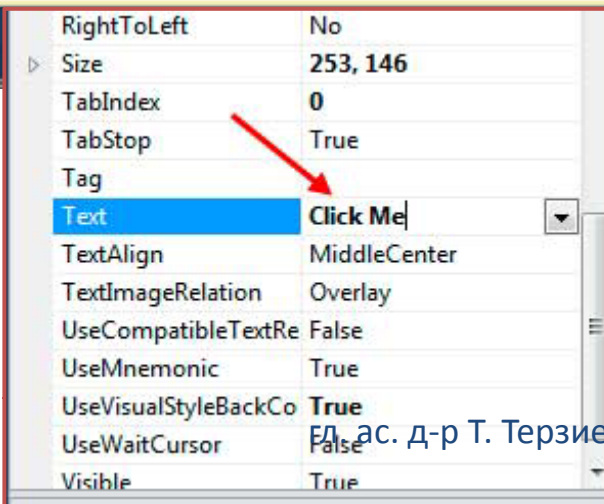
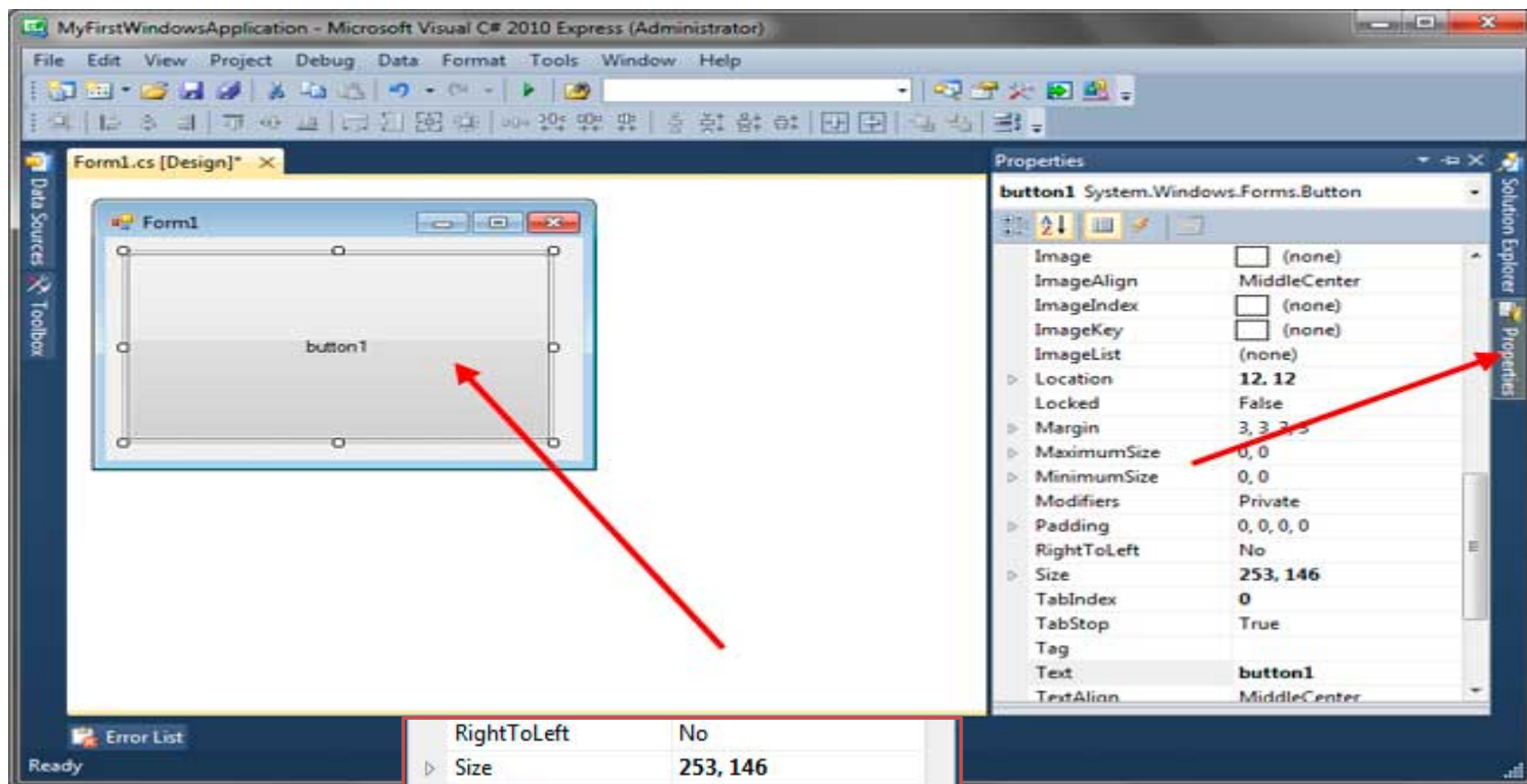
## ❖ Добавяне на контрола



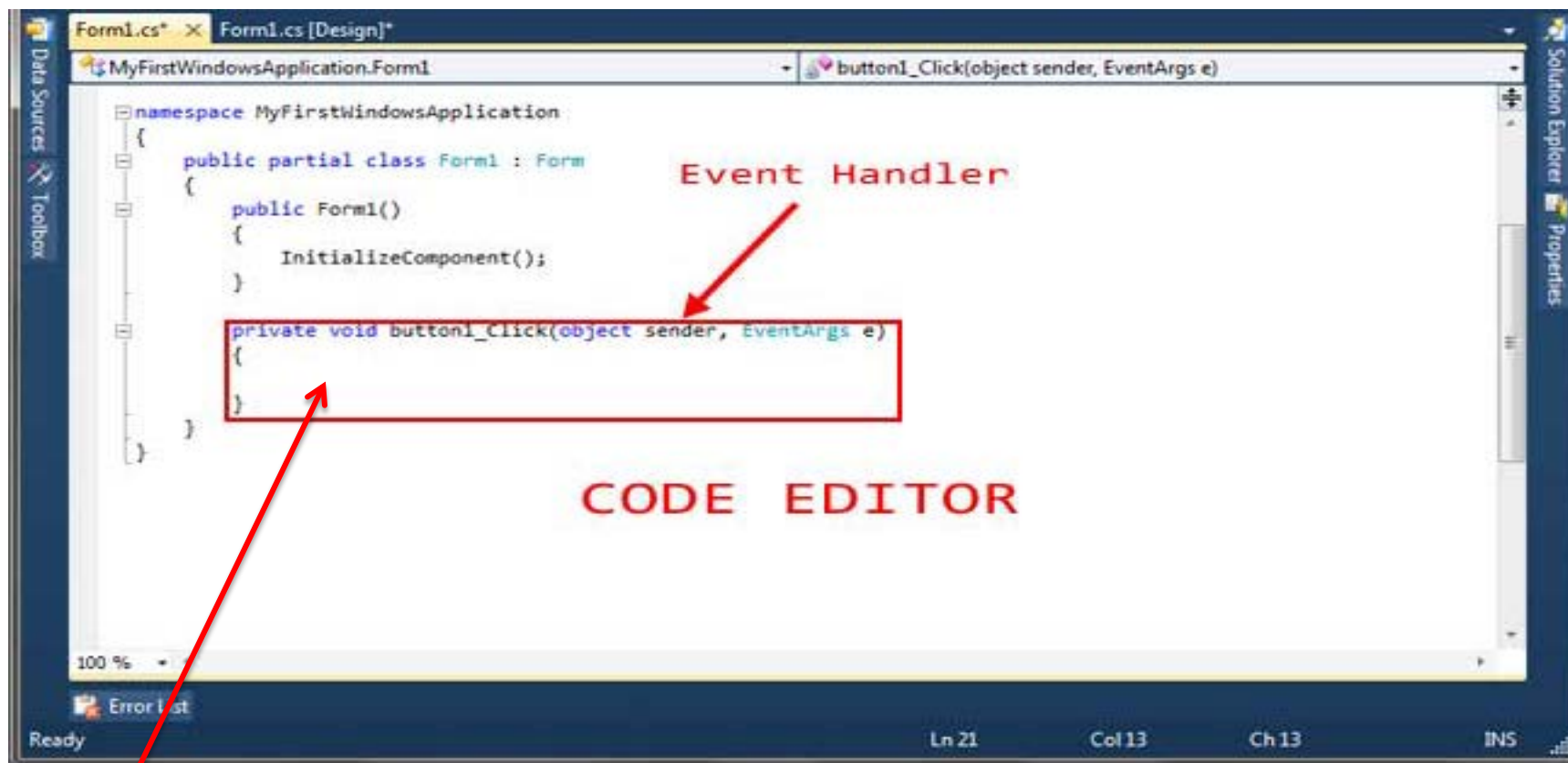
## ❖ Добавяне на неграфични компоненти



# Настройка на свойства (Properties)



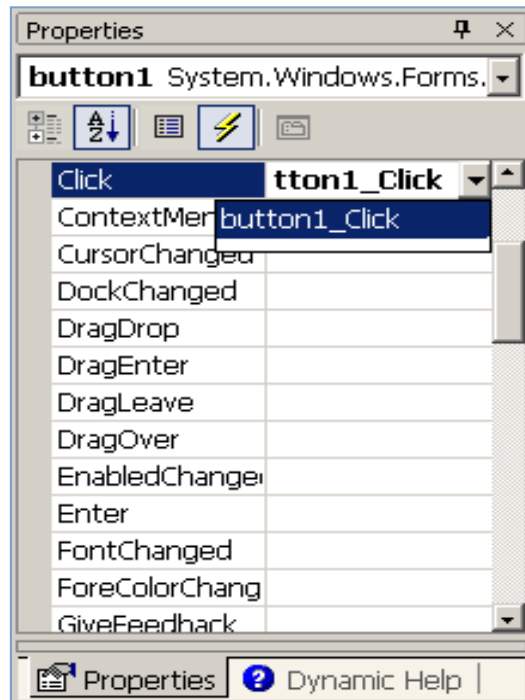
# Добавяне на обработчици на събития (Event Handlers)



```
private void button1_Click(object sender, EventArgs e) {
    MessageBox.Show("You clicked the button!"); }
```



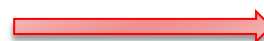
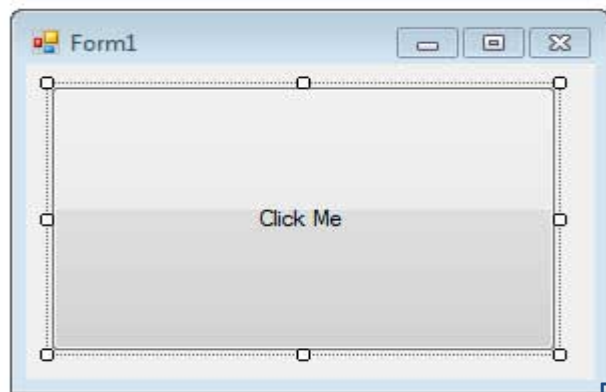
# Добавяне на обработчици на събития (Event Handlers)



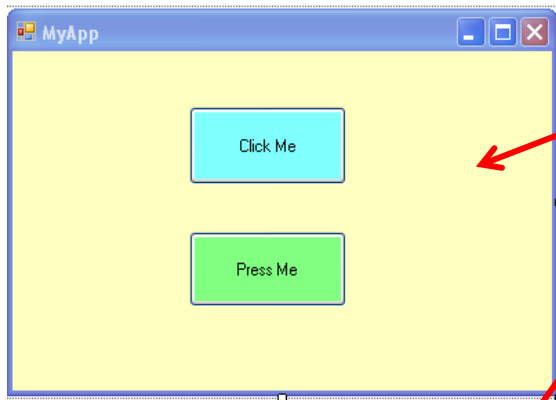
```
private void button1_Click(object sender, System.EventArgs e)
{
    // Code to be executed when the button is clicked
}
```

The code editor shows the implementation of the `button1_Click` event handler. A red arrow points from the `button1_Click` handler name in the Properties window to the start of the code block.

```
{ MessageBox.Show("You clicked the button!"); }
```



# Пример 1: Създаване на приложение с два командни бутона и един етикет



1. Създаване на ГПИ на приложението – добавяне на етикет и бутони
2. Настройка на някои свойства на елементите на ГПИ в режим на проектиране
3. Добавяне на програмен код към елементите

Елементи на ГПИ	Име на елемент	Свойства
Label	<b>label1</b>	Text="Здравейте колеги!"; Font, Bold, 14, BackColor – по избор
Button	<b>button1</b>	Text="Click me "; Font, Bold, 14,
Button	<b>button2</b>	Text="Press me "; Font, Bold, 14, Visible = False
Form	<b>Form1</b>	StartPosition=CenterScreen Text="My App"

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Това е диалогова кутия!");
}
private void button2_Click(object sender, EventArgs e)
{
    MessageBox.Show("Създаване на ГПИ C#");
}
```

## Пример 2: Създаване на графично приложение, което събира две цели числа

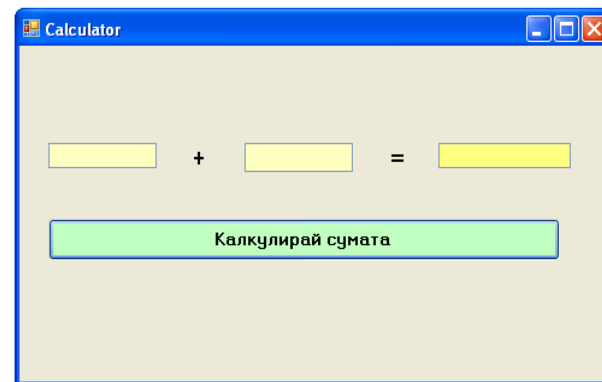
1. Стартираме VS.NET и създаваме нов **Windows Forms** проект.

2. Задаваме на **главната форма** име **Calculator**

3. Създаване на ГПИ на приложението:

От Toolbox на VS.NET добавяме три **TextBox**, две **Label** и една **Button** контроли

4. Настройка на някои свойства на елементите на ГПИ в режим на проектиране:



Елементи на ГПИ	Име на елемент	Свойства
Label	labelPlus	Text="+" ; Font, Bold, 14, BackColor – по избор
Label	labelEquals	Text="=" ; Font, Bold, 14, BackColor – по избор
TextBox	textBox1Number1	Text=" " ; Font, Bold, 14, BackColor – по избор
TextBox	textBox1Number2	Text=" " ; Font, Bold, 14, BackColor – по избор
TextBox	textBox3Sum	Text=" " ; Font, Bold, 14, BackColor – по избор <b>ReadOnly = true</b>
Button	buttonCal	Text="Калкулирай сумата" ;
Form	Form1	StartPosition=CenterScreen Text="Calculator"

# Добавяне на програмен код към елементите



```
private void button1CalcSum_Click(object sender, EventArgs e)
```

```
{  
    int sum, x, y;  
    x = int.Parse(textBox1Number1.Text);  
    y = int.Parse(textBox2Number2.Text);  
    sum = x+y;  
    textBox3Sum.Text = sum.ToString();  
}
```



```
private void ButtonCalcSum_Click(object sender, System.EventArgs e)
```

```
{ try  
    {  
        int x = int.Parse(textBox1Number1.Text);  
        int y = int.Parse(textBox2Number2.Text);  
        int sum = x + y;  
        textBox3Sum.Text = sum.ToString();  
    }  
    catch (FormatException)  
    {  
        TextBox3Sum.Text = "Невалидни данни!";  
    }  
  
    TextBoxNumber1.SelectAll();  
    TextBoxNumber2.SelectAll();  
    TextBoxNumber1.Focus()  
}
```

Благодаря за вниманието!

