



ASSOSOFTWARE

ASSOCIAZIONE ITALIANA PRODUTTORI SOFTWARE

.NET Core in C#

Marchetti Filippo

14/12/2023

SOFTWARE HUB
system srl



.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

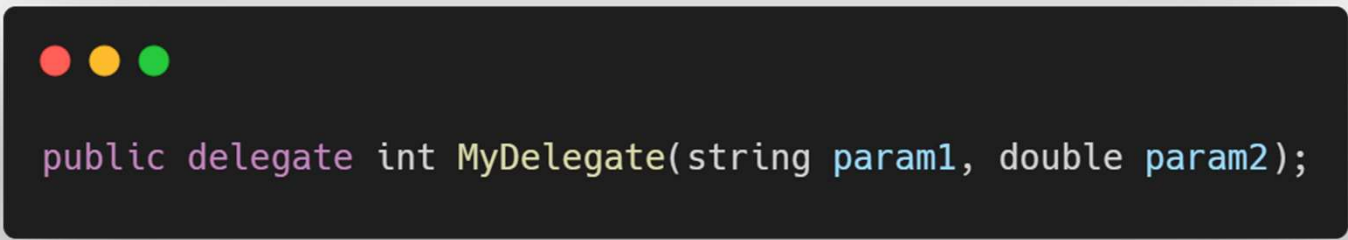
- Delegate ed Eventi
- Sviluppare applicazioni con Windows Forms
- Introduzione a XAML
- Sviluppare applicazioni con WPF

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Delegate ed Eventi

- L'ambiente di esecuzione .NET usa meccanismi di **notifica** per consentire a oggetti di classi differenti di **comunicare fra di loro** e inviarsi i **messaggi** al verificarsi di particolari **eventi**.
- L'implementazione di tale modello si basa sui **delegate**.
- I delegate sono dei *reference type* ma, anziché referenziare, ad esempio, un'istanza di una classe, rappresenta un **metodo**. In questo modo è possibile trattare i metodi come se fossero dati, assegnandoli ad una variabile, passandoli come argomenti a un altro metodo e così via.



```
public delegate int MyDelegate(string param1, double param2);
```

- Ogni delegate è **type-safe**, cioè sicuro rispetto ai tipi che può trattare, per definizione.

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Delegate ed Eventi

- **Creare un delegate.** Una possibilità per istanziare un delegate è quella che prevede l'utilizzo di new, indicando il nome del metodo come se si trattasse del parametro di un normale costruttore.

```
public delegate string IntToStringDelegate(int number);

public class Converter
{
    public string ConvertToString(int intVar)
    {
        return intVar.ToString();
    }
}

...

IntToStringDelegate i2sDelegate = new IntToStringDelegate(ConvertToString);
```


.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Delegate ed Eventi

- **Invocare un delegate.** E' possibile invocare un delegate, richiamandolo direttamente (passando eventuali parametri in input) oppure invocando esplicitamente il metodo **Invoke** sull'istanza.

```
string stringVar1 = i2sDelegate(123);  
  
...  
  
string stringVar2 = i2sDelegate.Invoke(123);  
  
...  
  
public void UserDelegate(IntToStringDelegate myDelegate, params int[] values)  
{  
    foreach(int @value in values)  
    {  
        myDelegate(@value);  
    }  
}
```

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Delegate ed Eventi

- **Delegate multicast.** I delegate risultano particolarmente utili in quanto possono mantenere al loro interno uno o più riferimenti a diversi metodi (ovviamente rispettando la sua firma). Tale possibilità è particolarmente utile per scrivere codice che possa notificare più oggetti invocando in ognuno un metodo.

```
public delegate void EmptyDelegate();  
  
...  
  
public void Method1() { Console.WriteLine("Metodo 1"); }  
  
...  
  
public void Method2() { Console.WriteLine("Metodo 2"); }  
  
...  
  
EmptyDelegate multicastDelegate = Method1;  
multicastDelegate += Method2;  
  
...  
  
multicastDelegate(); // Invoco il delegate multicast
```

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Delegate ed Eventi

- Ora che abbiamo appreso il concetto di delegate, è possibile affrontare l'argomento che ne costituisce una delle principali applicazioni pratiche, vale a dire gli **eventi**.
- Consentono ad un oggetto di avvisare gli altri oggetti che si è verificato qualcosa che potrebbe interessarli. Sono uno dei principali meccanismi utilizzati all'interno di applicazioni con UI.
- L'oggetto che genera l'evento è detto **publisher** mentre quelli che restano in ascolti dell'evento specifico, ed eventualmente lo gestiscono, sono detti **subscriber**.
- Questo pattern di programmazione è detto *publish-subscribe*, oppure produttore-consumatore.
- La parola chiave **event** di C# consente di definire un particolare tipo di membro di una classe, detto evento.

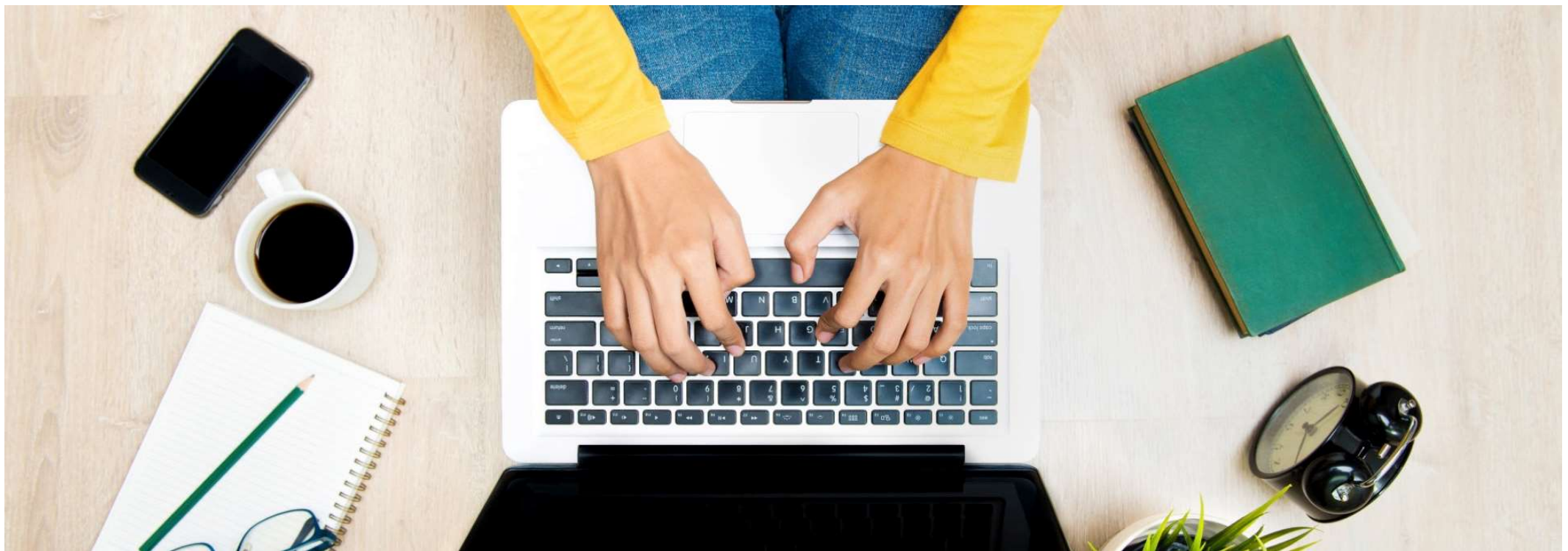
```
public class Car
{
    public event EventHandler EngineOverRevved;
    public event EventHandler EngineOff;

    ...
}
```

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Delegate ed Eventi

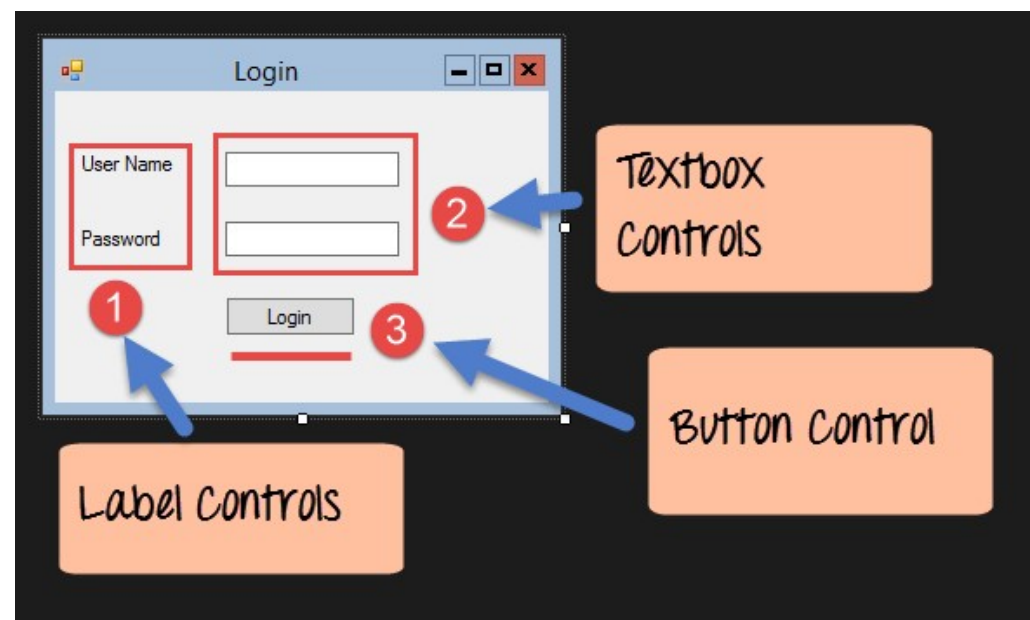


.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Sviluppare applicazioni con Windows Forms

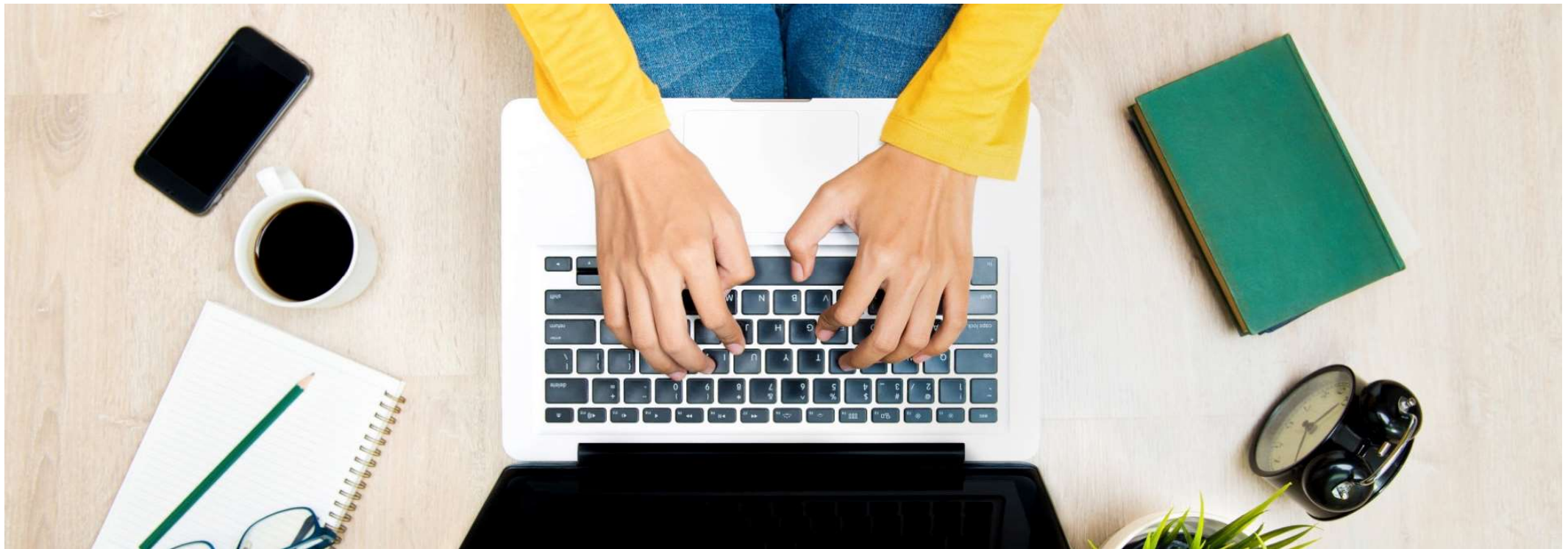
- E' la prima modalità (ormai «datata») messa a disposizione dal framework .NET per lo sviluppo di interfacce grafiche per ambienti Windows.
- Esistono diversi tipi di controlli disponibili per Windows Forms. Alcuni dei controlli più comuni includono:
 - *Button*. Consentono agli utenti di eseguire azioni, come aprire un file o salvare le modifiche.
 - *TextBox*: Permettono agli utenti di inserire del testo.
 - *ListBox*: Consentono agli utenti di visualizzare e selezionare un elenco di elementi.
 - *ComboBox*: Consentono agli utenti di visualizzare e selezionare un elemento da un elenco.
- Il namespace che contiene tutti gli strumenti per lo sviluppo è *System.Windows.Forms*
- Un'applicazione Windows Form è un'applicazione basata su **eventi** supportata da .NET Framework di Microsoft. A differenza di un programma batch, trascorre la maggior parte del tempo semplicemente aspettando che l'utente faccia qualcosa, come riempire una casella di testo o fare clic su un pulsante.



.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Sviluppare applicazioni con Windows Forms




.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)


Introduzione a XAML

- **XAML** (*eXtensible Application Markup Language*), è un linguaggio di markup basato su XML (*eXtensible Markup Language*) utilizzato principalmente per definire interfacce utente per applicazioni Microsoft, in particolare nelle applicazioni *Windows Presentation Foundation* (WPF), *MAUI*, *Xamarin* e in alcune parti delle app *Universal Windows Platform* (UWP).
- XAML segue una struttura gerarchica basata su tag. Gli elementi sono organizzati in una struttura ad albero. Ogni elemento può contenere attributi e altri elementi figlio.



```
<Elemento Attributo="Valore">Contenuto</Elemento>
```

- Un documento XAML è costituito da un insieme di elementi XML. Ogni elemento XAML rappresenta un **oggetto grafico**, come un pulsante, un'etichetta o un controllo di testo.



```
<Button x:Name="myButton" Content="Ciao mondo!">  
</Button>
```

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

- **La sintassi Object Element.** Questo tipo di sintassi rappresenta un'istanza di un oggetto; ogni elemento è formato da una parentesi angolare aperta "<" il nome della classe da istanziare, seguito da una slash e da una parentesi angolare chiusa ">".



```
<Button>  
Hello!  
</Button>
```

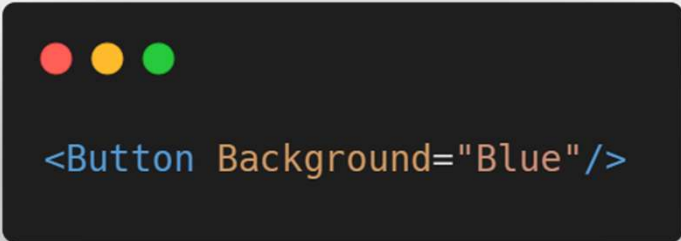
- Nel frammento di codice precedente, l'inserimento della semplice stringa di testo "hello" è equivalente a impostare la proprietà Content della classe Button.

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

- **La sintassi Property Attribute.** Possiamo impostare le proprietà di un oggetto utilizzando il nome della proprietà come attributo di un elemento. Nell'esempio sotto impostiamo il colore di sfondo di un button. Quindi il nome dell'attributo rappresenta il nome della proprietà e la stringa dopo il simbolo dell'uguale ne rappresenta il valore.



```
<Button Background="Blue" />
```

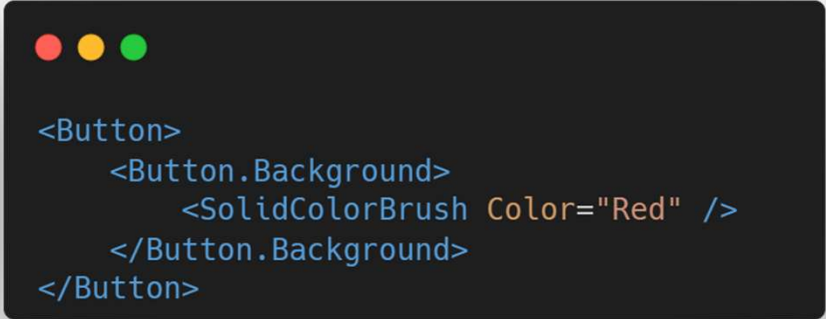
- Ci sono proprietà che comunque possono difficilmente essere rappresentate da una semplice stringa. In questi casi possiamo ricorrere a una sintassi alternativa, chiamata *Property Element*.

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

- **La sintassi Property Element.** In questo tipo di sintassi, il valore della proprietà è espresso utilizzando l'*Object Element*, mentre la proprietà da impostare segue la sintassi *Type.PropertyName* e prende forma come elemento interno all'oggetto che la espone.



```
<Button>  
    <Button.Background>  
        <SolidColorBrush Color="Red" />  
    </Button.Background>  
</Button>
```

- Nel precedente esempio creiamo un'istanza di una classe *Button* e impostiamo la proprietà *Background*, utilizzando una nuova istanza di *SolidColorBrush*.

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

➤ Il layout system.

- Ogni elemento dell'interfaccia occupa uno spazio che è chiamato **bounding box**, il quale viene definito dal layout system: è un processo ricorsivo che misura, arrangia, dispone e si occupa della renderizzazione.
- Quando in un documento XAML, attraverso il markup, definiamo gli elementi dell'interfaccia, andiamo a delineare, a tutti gli effetti, una gerarchia ad albero, che viene chiamato **Logical Tree**.
- Ogni elemento logico può essere formato da uno o più elementi grafici. Per esempio, il componente *Button* è costituito da più elementi visuali, e questa gerarchia prende il nome di **Visual Tree**.

```
DUMPING VISUAL TREE:
Original Element: ButtonChrome
Closest Visual Ancestor to Original Element: <self>
TemplatedParent of Closest Visual Ancestor: Button

0> Window
1> Border
2> AdornerDecorator
3> ContentPresenter
4> StackPanel
5> TextBlock
6> ContainerVisual
7> ComboBox
8> Grid
9> Popup
9> ToggleButton
10> ButtonChrome
11> Grid
12> Path
9> ContentPresenter
10> TextBlock
5> Button
6> ButtonChrome [YOU CLICKED HERE]
7> ContentPresenter
8> TextBlock
5> ListBox
6> Border
7> ScrollViewer
8> Grid
9> Rectangle
9> ScrollContentPresenter
10> ItemsPresenter
11> VirtualizingStackPanel
12> ListBoxItem
13> Border
14> ContentPresenter
15> TextBlock
12> ListBoxItem
13> Border
14> ContentPresenter
15> TextBlock
12> ListBoxItem
13> Border
14> ContentPresenter
15> TextBlock
12> ListBoxItem
13> Border
14> ContentPresenter
15> Border
16> CheckBox
17> BulletDecorator
18> BulletChrome
18> ContentPresenter
19> TextBlock
10> AdornerLayer
9> ScrollBar
9> ScrollBar
3> AdornerLayer
4> FocusVisualAdorner
5> Control
6> Rectangle
*****
```

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

➤ Il layout system.

- **I pannelli.** La classe *Panel* è una classe astratta che fornisce l'infrastruttura per la realizzazione di classi specializzate per il posizionamento degli elementi dell'interfaccia utente. A questo scopo, esiste una serie di pannelli che risponde alle più semplici esigenze:
 - *Canvas*: è il più semplice dei pannelli. Gli elementi figlio sono disposti, mediante coordinate assolute, relativamente a una coppia di margini, attraverso le *AttachedProperty Left, Top, Right e Bottom*.
 - *Grid*: gli elementi sono disposti in righe e colonne.
 - *StackPanel*: gli elementi sono disposti l'uno di seguito all'altro, verticalmente oppure orizzontalmente, in base alle proprietà *Orientation*.
 - *DockPanel*: semplifica la disposizione degli elementi figlio all'interno di un contenitore in base a direttive di ancoraggio (docking – occupazione dello spazio rimanente).
 - *VirtualizingStackPanel*: la disposizione degli elementi è identica a quella eseguita dallo *StackPanel* ma è ottimizzata per un numero elevato di elementi, calcolando solamente quelli effettivamente visibili.

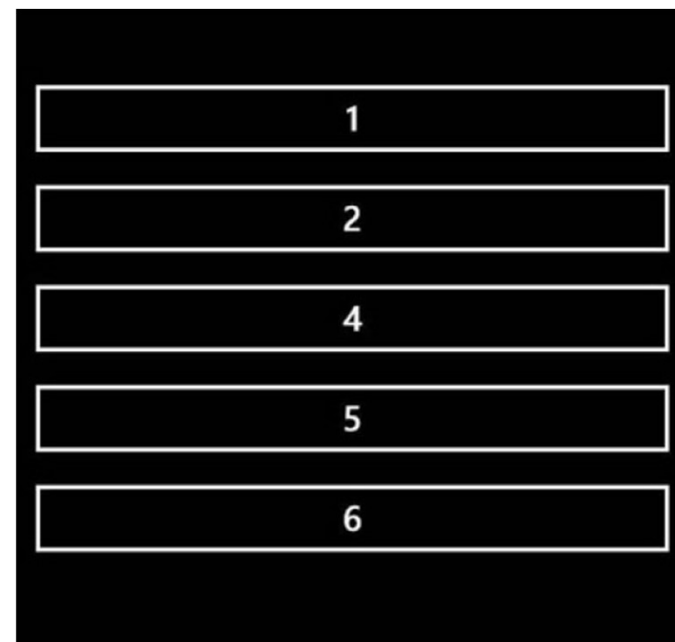
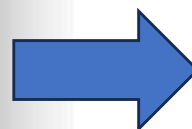
.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

➤ Il layout system.

```
<UserControl
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markupcompatibility/2006"
  Width="300"
  Height="300">
  <StackPanel>
    <Button>1</Button>
    <Button>2</Button>
    <Button>4</Button>
    <Button>5</Button>
    <Button>6</Button>
  </StackPanel>
</ UserControl >
```



.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

➤ I controlli.

- Dovendo dividere i controlli, possiamo distinguerli in due grandi categorie: i controlli che derivano direttamente o indirettamente dalla classe *ContentControl* e quelli che derivano da *ItemsControl*. Un esempio di classe derivata dal tipo *ContentControl* è il controllo *Button*.
- Tutti i controlli che derivano dalla classe *ContentControl* espongono la proprietà **Content** di tipo *Object*. Ciò permette, dal semplice *Button* allo *UserControl*, di ospitare qualsiasi tipo di contenuto, dalla semplice stringa di testo fino a un complesso Logical Tree.



```
<Button>
  <Image Height="50"
    Source="Desert.jpg"
    Stretch="Fill" />
</Button>
```


.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

➤ I controlli.

- Il tipo *ItemsControl* espone il proprio contenuto attraverso la proprietà **Items** ed è la classe base per una serie di controlli, come la *ListBox*, un controllo utile per gestire la visualizzazione e la selezione di uno o più elementi. La capacità di selezionare non è fornita dalla classe *ItemsControl* ma da *Selector*, uno dei suoi tipi derivati.

```
<ListBox>
  <Image Height="80"
    Source="3.1.png"
    Stretch="None" />
  <Image Height="80"
    Source="xp.png"
    Stretch="none" />
  <Image Height="80"
    Source="vista.png"
    Stretch="none" />
  <Image Height="80"
    Source="8.png"
    Stretch="none" />
</ListBox>
```

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

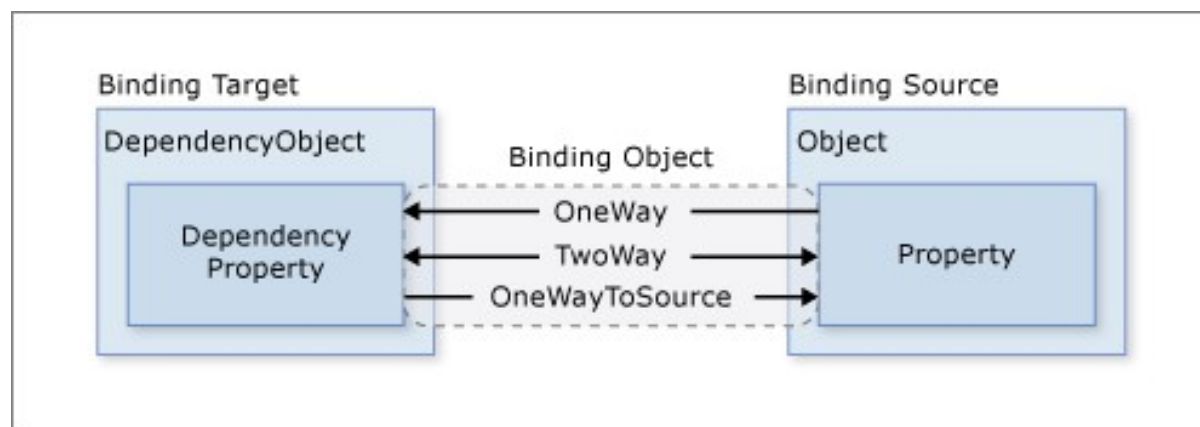
Introduzione a XAML

Data Binding. Nella maggior parte delle applicazioni, lo scopo principale è mostrare dati provenienti da database, servizi, oppure dal web, per permetterne la *lettura*, la *modifica*, la *cancellazione* o l'*inserimento*. Poiché questa esigenza è molto frequente e presenta spesso le medesime dinamiche, in XAML esiste un meccanismo che ci facilita queste operazioni: il *data binding*.

Con questo termine si indica un **legame** che viene creato tra **controlli** e **sorgente dati**, in modo da riflettere le modifiche apportate al controllo sulla sorgente, e viceversa.

Il data binding ha tre componenti

- *Binding source*
- *Binding target*
- *Binding object*

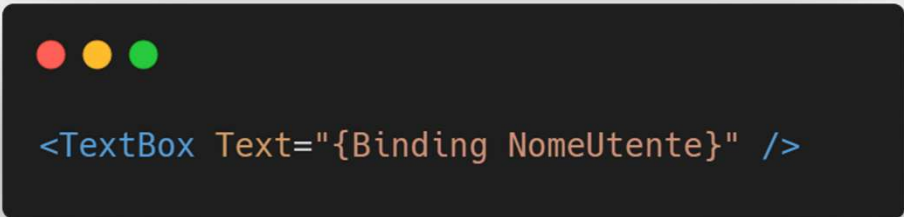


.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

- **Binding Source.** Rappresenta l'origine dei dati. È l'oggetto o la proprietà dai quali si desidera ottenere i dati da visualizzare o con cui si desidera effettuare un'operazione di aggiornamento.
- **Binding Target.** Rappresenta la destinazione dei dati. È l'elemento dell'interfaccia utente o la proprietà a cui si desidera collegare i dati. In genere, è un controllo dell'interfaccia utente come un *TextBox*, un *TextBlock*, un'immagine, ecc.
- **Binding Object.** E' l'oggetto che rappresenta la connessione tra il Binding Source e il Binding Target. Contiene informazioni su come i dati dovrebbero fluire tra la sorgente e la destinazione.



```
<TextBox Text="{Binding NomeUtente}" />
```

- In questo esempio:
 - *NomeUtente* è la proprietà del *Binding Source* (ad esempio, un oggetto *User*).
 - Il *TextBox* è il *Binding Target*.
 - Il *Binding Object* viene creato implicitamente per gestire la connessione tra la sorgente e la destinazione.

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Introduzione a XAML

Le modalità di data binding. Specificano come i dati devono fluire tra la sorgente e la destinazione. Le modalità di data binding principali includono:

- **OneWay** (*unidirezionale*). I dati vengono trasmessi solo dal *source* al *target*. Qualsiasi modifica nel source si riflette nel target, ma le modifiche nel target non influiscono sulla sorgente.
- **TwoWay** (*bidirezionale*). I dati possono fluire sia dal source al target che dal target al source. Le modifiche nella sorgente influenzano il target e viceversa.
- **OneTime** (*una volta*). I dati vengono trasmessi solo una volta dal source al target durante l'inizializzazione. Eventuali modifiche successive nella sorgente non influiscono sulla destinazione.
- **OneWayToSource** (*unidirezionale verso il source*). I dati fluiscono solo dal target verso il source. Le modifiche nel target influiscono sulla sorgente, ma non viceversa.

```
<TextBox Text="{Binding Nome, Mode=TwoWay}" />
```

.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Sviluppare applicazioni con WPF



- **WPF**, acronimo di *Windows Presentation Foundation*, rappresenta un framework per la creazione di interfacce utente impiegato nella realizzazione di applicazioni destinate all'ambiente desktop di Windows. Presente nel framework .NET dalla versione 3.0, offre un approccio per sviluppare interfacce utente che siano coinvolgenti e interattive mediante l'utilizzo di **XAML**.

- Alcune delle caratteristiche di WPF:
 - **Grafica vettoriale.** WPF sfrutta la grafica vettoriale, consentendo la creazione di elementi grafici scalabili senza perdita di qualità.
 - **Data binding.** WPF semplifica il data binding, consentendo la connessione diretta tra i dati e gli elementi dell'interfaccia utente.
 - **Animazioni e trasformazioni.** WPF supporta animazioni fluide e trasformazioni, consentendo agli sviluppatori di creare UI interattive e dinamiche.
 - **3D Graphics:** WPF offre anche il supporto per la grafica tridimensionale
 - **Supporto multimediale:** WPF integra funzionalità multimediali, facilitando l'integrazione di audio, video e altri contenuti multimediali nelle applicazioni.
 - **Separazione UI e logica:** WPF favorisce la separazione tra la logica dell'applicazione e la presentazione dell'interfaccia utente utilizzando il *pattern MVVM*.

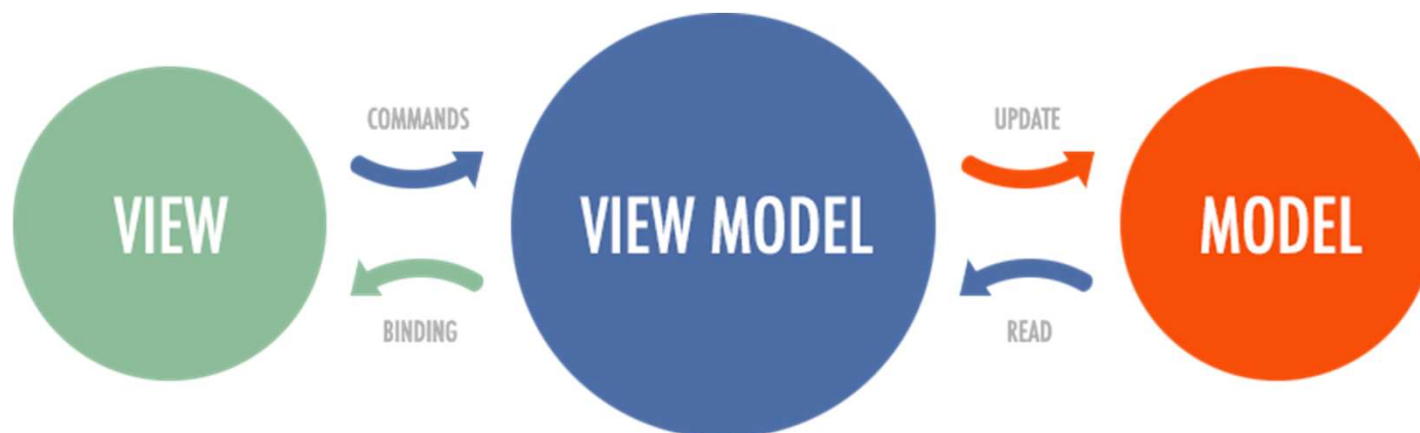
.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Sviluppare applicazioni con WPF

Il pattern MVVM. MVVM sta per *Model-View-ViewModel* ed è un design pattern ampiamente utilizzato nelle applicazioni WPF per separare la logica di presentazione dalla logica di business.

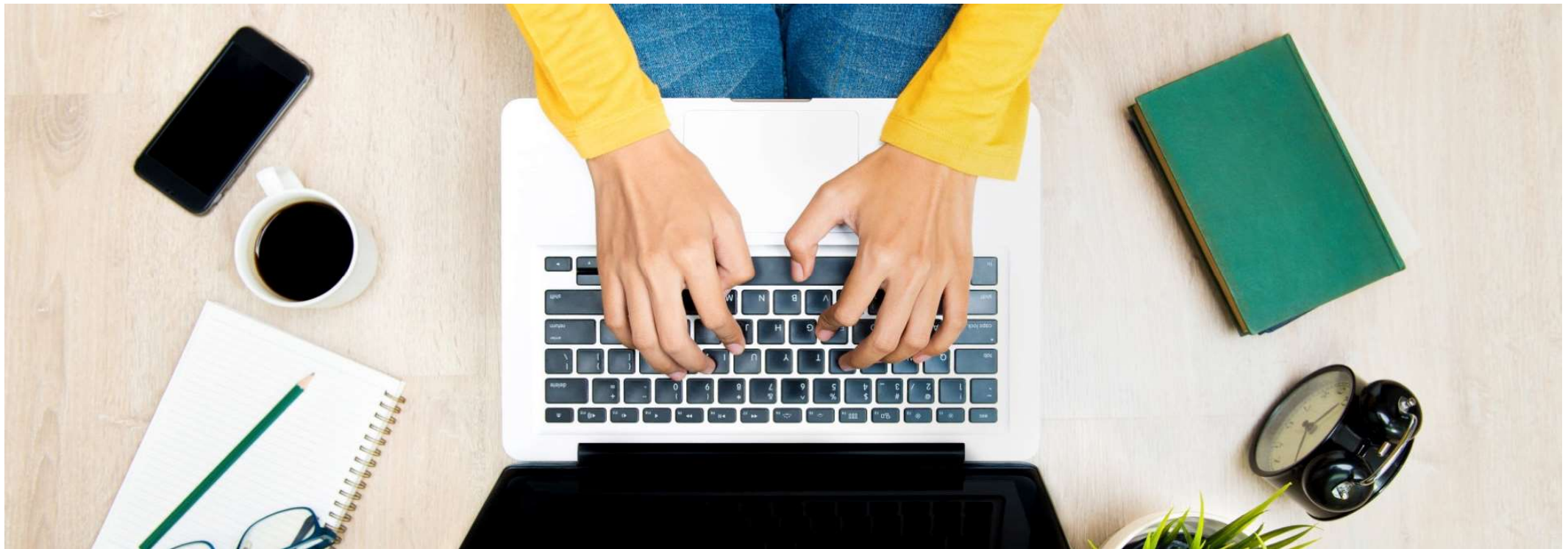
- **Model (modello).** Rappresenta la **logica di business dell'applicazione**, gestendo i dati, le regole di business e le operazioni di accesso ai dati. Non è consapevole della UI o delle operazioni di presentazione.
- **View (vista).** E' responsabile della presentazione dell'interfaccia utente (UI) e dell'interazione con l'utente. Non contiene logica di business; si limita a mostrare i dati forniti dal *ViewModel* e/o a inoltrare gli input dell'utente al *ViewModel*.
- **ViewModel.** Funge da **intermediario** tra la View e il Model. Si occupa di esporre i **dati** e i **comandi** necessari per la *View*, senza essere direttamente vincolato alla struttura della UI. Converte i dati del *Model* in una forma comprensibile dalla *View* e gestisce le interazioni utente invocando i comandi appropriati nel *Model*. Implementa spesso l'**INotifyPropertyChanged** per notificare la *View* quando i dati cambiano.



.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)

Sviluppare applicazioni con WPF



.NET CORE IN C#

Lezione 6: Creazione di Interfacce Utente (UI)



<https://www.menti.com>

1160 1731