Управління ресурсами та прив'язка даних в технології WPF

Інструментальні засоби візуального програмування

Лекція 04, ЧДБК-2019

План лекції

- Поняття ресурсів у WPF-додатках
- Архітектура MVVM та технологія прив'язування даних
- Властивості залежності: детальніший огляд
- Декорування вбудованих елементів управління
- Робота зі стилями, шаблонами та тригерами

Поняття ресурсів у WPF-додатках

Питання 4.1.

Поняття ресурсу

- Традиційні ресурси додатку складаються з бінарних порцій даних у формі іконок, бітових карт, рядків, XML та ін.
  - .NET framework постачає узагальнену підтримку за допомогою класу ResourceManager.
- Технологія WPF вводить поняття ***логічних ресурсів***.
  - Це будь-які об'єкти, які можна використовувати та поширювати в багатьох місцях додатку, навіть між збірками.
  - Деякі з можливостей WPF (implicit (automatic) styles, type-based data templates) засновані на системі логічних ресурсів.
- Можуть бути різними: від WPF-об'єктів (brushes, geometries, styles) до інших об'єктів .NET Framework або розробника (string, List<> або власно створений об'єкт).
  - Такі об'єкти зазвичай розташовуються всередині ResourceDictionary та розміщуються в пам'яті під час виконання програми за допомогою ієрархічного пошуку.

Логічні ресурси

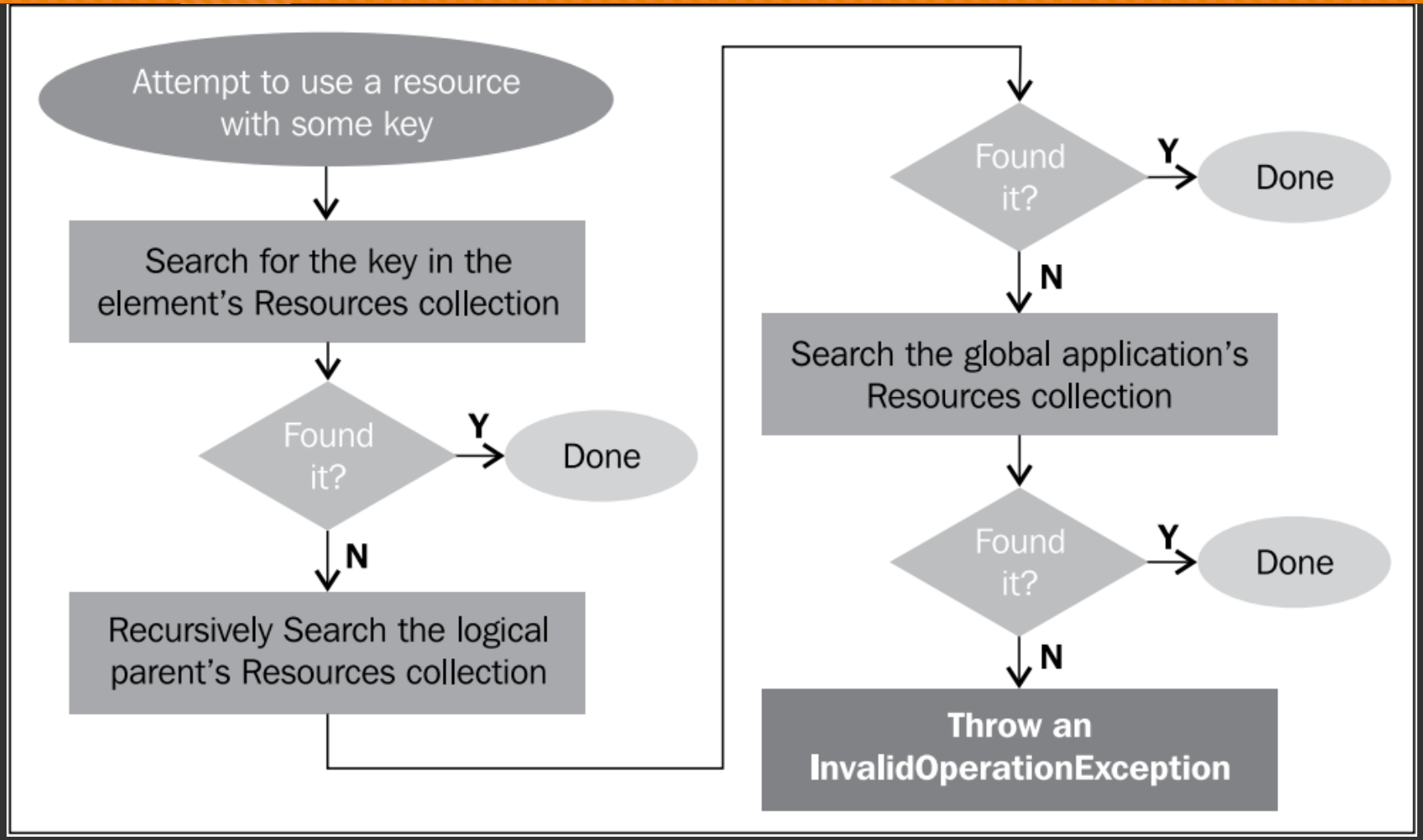
- Кожен елемент, породжений від FrameworkElement, має властивість Resources типу ResourceDictionary.
  - У XAML у більшості випадків необхідно задавати атрибут x:Key (існують виключення у контексті стилів та шаблонів даних).
- Вбудована колекція елементу Window:

```
<Window.Resources>
    <LinearGradientBrush x:Key="brush1">
        <GradientStop Offset="0" Color="Yellow" />
        <GradientStop Offset="1" Color="Brown" />
    </LinearGradientBrush>
</Window.Resources>
```

- Використання цього ресурсу XAML вимагає розширення розмітки StaticResource:

```
<Rectangle Fill="{StaticResource brush1}" />
```

Алгоритм пошуку ресурсу в логічному дереві



Реалізація пошуку в коді

- Використовується метод FrameworkElement.FindResource():

```
Brush brush = (Brush)x.FindResource("brush1");
```

  - Ефект аналогічний до використання {StaticResource }.
- Перехоплення виключень, пов'язаних з неіснуючими ресурсами, можливе за допомогою методу TryFindResource():

```
Brush brush = (Brush)x.TryFindResource("brush1");
if(brush == null) { // not found }
```

- Можливий прямий доступ до ресурсу через індексатор.
  - У прикладі визначений пензель (brush) у ресурсах елементу Window:

```
Brush brush = (Brush)this.Resources["brush1"];
```

CH02.SimpleResources

- CH02.SimpleResources

Динамічне додавання та видалення ресурсів

- Словником Resources можна управляти під час виконання програми, додаючи та видаляючи ресурси:

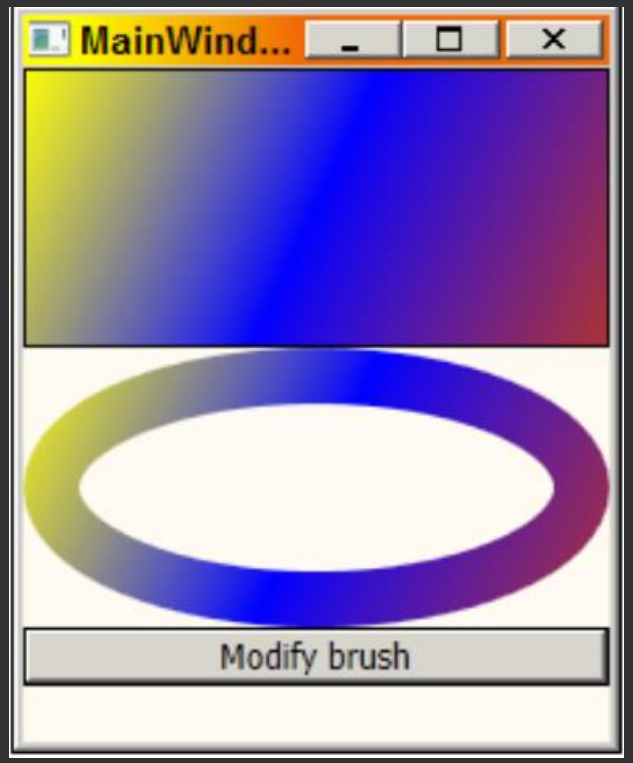
```
this.Resources.Add("brush2", new SolidColorBrush(
                                Color.FromRgb(200, 10, 150)));
```

  - Подальший виклик FindResource() допоможе локалізувати ресурс.

```
this.Resources.Remove("brush1");
```

  - Прив'язаний за допомогою {StaticResource} ресурс не втрачається, на нього можна посилатись.
  - Проте подальші виклики FindResource() до нього будуть невдалі.

Внесення змін до ресурсів



- Усі {StaticResource} (або FindResource) пошуки за ключем використовують the same object instance.
  - This means that modifying the resource properties (not replacing the resource with a different one), impacts automatically all properties using that resource.
  - Наприклад, зміни в ресурсі-пензлі:

```
var brush =
    (LinearGradientBrush)this.Resources["brush1"];
brush.GradientStops.Add(new
    GradientStop(Colors.Blue, .5));
```

  - Призводить до негайних змін у виводі.

Використання ресурсів іншими ресурсами

```
<LinearGradientBrush x:Key="brush3">
        <GradientStop Offset="0" Color="Red" />
        <GradientStop Offset="1" Color="Orange" />
</LinearGradientBrush>

<DataTemplate x:Key="temp1">
        <Rectangle      Fill="{StaticResource brush3}"
                        StrokeThickness="4" Stroke="DarkBlue" />
</DataTemplate>
```

Ресурси без спільного доступу (Non-shared resources)

- За замовчуванням доступ до ресурсів загальний – вони існують в одному екземплярі.
  - Якщо потрібно створювати нові екземпляри конкретного ресурсу при кожному lookup, використовується атрибут x:Shared="False" у визначенні ресурсу.

Ресурси в інших місцях розташування

- Не тільки елементи мають властивість Resources (ResourceDictionary).
  - Поширений приклад – template types (породжені від FrameworkTemplate): DataTemplate, ControlTemplate та ItemsPanelTemplate.
  - Вони можуть мати ресурси, що будуть доступні при їх використанні.

Динамічна прив'язка (binding) до логічного ресурсу

- Що відбувається при заміні деякого статично прив'язаного (StaticResource) ресурсу?
  - Чи вплине це на всі об'єкти, які використовують цей ресурс?
  - Якщо ні, то як прив'язати ресурс динамічно?
- Замінимо StaticResource на DynamicResource
  - ***CH02.DynamicVsStatic***
  - Розширення розмітки DynamicResource виконує динамічну прив'язку до ресурсу: при заміні об'єкта на новий (з тим же ключем) використовується новий ресурс.
  - StaticResource змушує посилатись на старий об'єкт.

StaticResource vs DynamicResource

- StaticResource виконує прив'язку до ресурсу при конструюванні (у виклику InitializeComponent).
  - StaticResource рекомендують використовувати повсякчас, крім випадків заміни ресурсів «на льоту».
- DynamicResource виконує прив'язку тільки за потреби, що дещо пришвидшує завантаження додатку.
  - DynamicResource не викидає виключення, якщо ключа не існує.
  - Якщо цей ключ з'явиться пізніше, ресурс прив'яжеться коректно.

Використання користувацьких кольорів та шрифтів

- Ключ ресурсу є частиною entry в ResourceDictionary.
  - Фактично він типізується як object.
  - Приклад: ключ, який відповідає DynamicResource, є статичною властивістю SystemColors.DesktopBrushKey:
  - Fill="{DynamicResource {x:Static SystemColors.DesktopBrushKey}}"
  - Тип даної властивості ідентифікується як ResourceKey, який є абстрактним класом.
  - Конкретний тип не важливий, він буде внутрішнім для збірки PresentationFramework.

Використання користувацьких кольорів та шрифтів

- Шрифти використовують ту ж ідею.
  - Існують статичні властивості, які закінчуються на "Key", для кожного налаштування, що користувач може змінити через діалог (Персоналізація) Personalization.
  - Схожий клас SystemParameters містить набір інших загальних властивостей (з суфіксом "Key"), які можна змінювати (деякі через вікно Персоналізація, деякі – тільки в коді).
- Приклад: SystemParameters.CaptionHeightKey – визначає поточну стандартну висоту заголовка вікна.
  - Може допомогти при конструюванні власного шаблону для нестандартного вікна;

- Згадані класи SystemColors, SystemFonts та SystemParameters розкривають поточні значення різних властивостей у вигляді звичайних статичних властивостей.
  - Тому спрацює, наприклад, отримання поточного кольору пензля of the desktop: **Fill="{x:Static SystemColors.DesktopBrush}"**
- Робота проста, проте виконується не динамічно.
  - Значення зчитується під час виконання (парсингу XAML, зазвичай при конструюванні вікна), проте залишається тим же, поки це вікно не буде заново створене (зазвичай при перезапуску додатку).
  - Схема проста та полегшена: WPF не повинен моніторити зміни кольору.
  - Працює в сценаріях, коли негайний відгук на зміни кольорів/шрифтів/розмірів не потребується.

Використання бінарних ресурсів

- Порції байтів, які мають смисл для додатку: зображення, файли шрифтів тощо.
  - CH02.BinaryResources
- Перший бінарний ресурс (файл зображення) зберігається як ресурс всередині скомпільованої збірки.
  - Було встановлено Build Action = Resource для зображення.
  - Конкретний файл зображення стає непотрібним при розгортанні додатку.
  - Такі ресурси є частиною збірки та зберігаються в ресурсі НазваЗбірки.g.resources.

Доступ до бінарного ресурсу

- Може бути виконано кількома способами:
  - 1) За відносним шляхом (приклад: Images/apple.png).
  - _image.Source = new BitmapImage(
    new Uri("Images/apple.png", UriKind.Relative));

Доступ до бінарного ресурсу

- Коли Build Action задано значення Content, ресурс не включається у збірку.
  - Це зручніше при частих змінах ресурсу (зокрема і дизайнером), та небажаності перезбирання проекту.
  - Якщо ресурс об'ємний, його також небажано вносити в збірку.
  - Це можливе завдяки тому, що WPF додає атрибут AssemblyAssociatedContentFile у збірку, задаючи назву файлу ресурсу.

```
[assembly: AssemblyConfiguration("")]
[assembly: CompilationRelaxations(8)]
[assembly: AssemblyAssociatedContentFile("images/jellyfish.jpg")]
[assembly: TargetFramework(".NETFramework,Version=v4.0,Profile=Client",
[assembly: AssemblyTitle("CH02.BinaryResources")]
[assembly: AssemblyCompany("")]
```

- Відносний URI насправді є shortcut на більш складну та повну схему URI, яку називають pack URI.
  - Відповідна розмітка: Source="Images/apple.png"
  - Еквівалент: Source="pack://application:,,,/Images/apple.png"
  - Це допомагає при відсутності доречних конвертерів типів.
- Схема URI типу «pack» запозичена з XML Paper Specifcation (XPS), а три коми are not optional values with some defaults, but rather escaped slashes.
  - Детальніше тут.

Доступ до бінарних ресурсів у коді

- Досить прямолінійний, проте працює для ***стандартних*** ресурсів.
  - Створимо додаток, що відображатиме інформацію про книги, програмно зчитану з XML-файлу.
  - ***CH02.BinaryResourcesInCode***
- Статичний метод Application.GetResourceStream() постачає програмний спосіб доступу до ресурсу, використовуючи відносний URI (або абсолютний зі схемою типу «pack»).
  - Повертає об'єкт класу StreamResourceInfo, який містить 2 властивості:
    - ContentType повертає тип MIME (image/jpeg, text/xml тощо)
    - Stream забезпечує доступ до власне бінарних даних.
  - Якщо ресурс був відміченим для Build Action із значенням Content, then the similar looking Application.GetContentStream method should be used instead.

Існує інший спосіб отримати ресурс

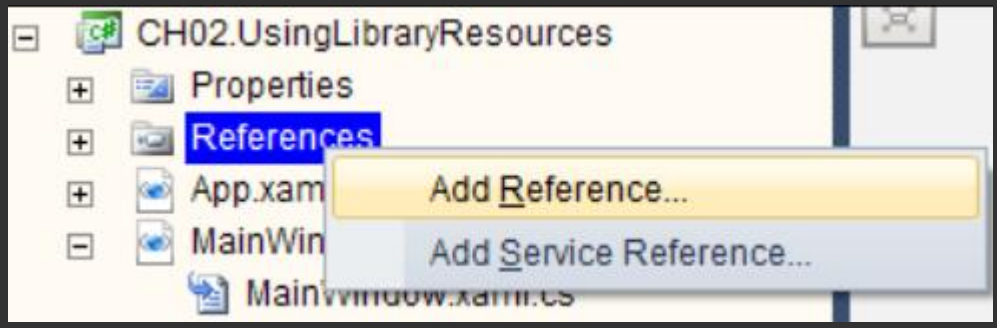
- Використовуючи базові .NET-типи, як ResourceManager та ResourceSet замість виклику Application.GetResourceStream.

```
Stream GetResourceStream(string name) {
   string asmName = Assembly.GetExecutingAssembly().GetName().Name;
   var rm = new ResourceManager(asmName + ".g",
       Assembly.GetExecutingAssembly());
         using(var set = rm.GetResourceSet(
            CultureInfo.CurrentCulture, true, true)) {
      return (Stream)set.GetObject(name, true);
   }
}
```

  - Note that there's no counterpart for a resource on which Build Action was set to Content.

Доступ до бінарних ресурсів з іншої збірки

- Інколи бінарні ресурси визначені в одній збірці (зазвичай бібліотеці класів), а потрібні в іншій.
  - WPF забезпечує прямий доступ до цих ресурсів, використовуючи pack URI scheme.
  - ***CH02.ClassLibraryResources***
  - We're not going to actually use any user controls, but this is a simple way to create a class library with WPF references already included.



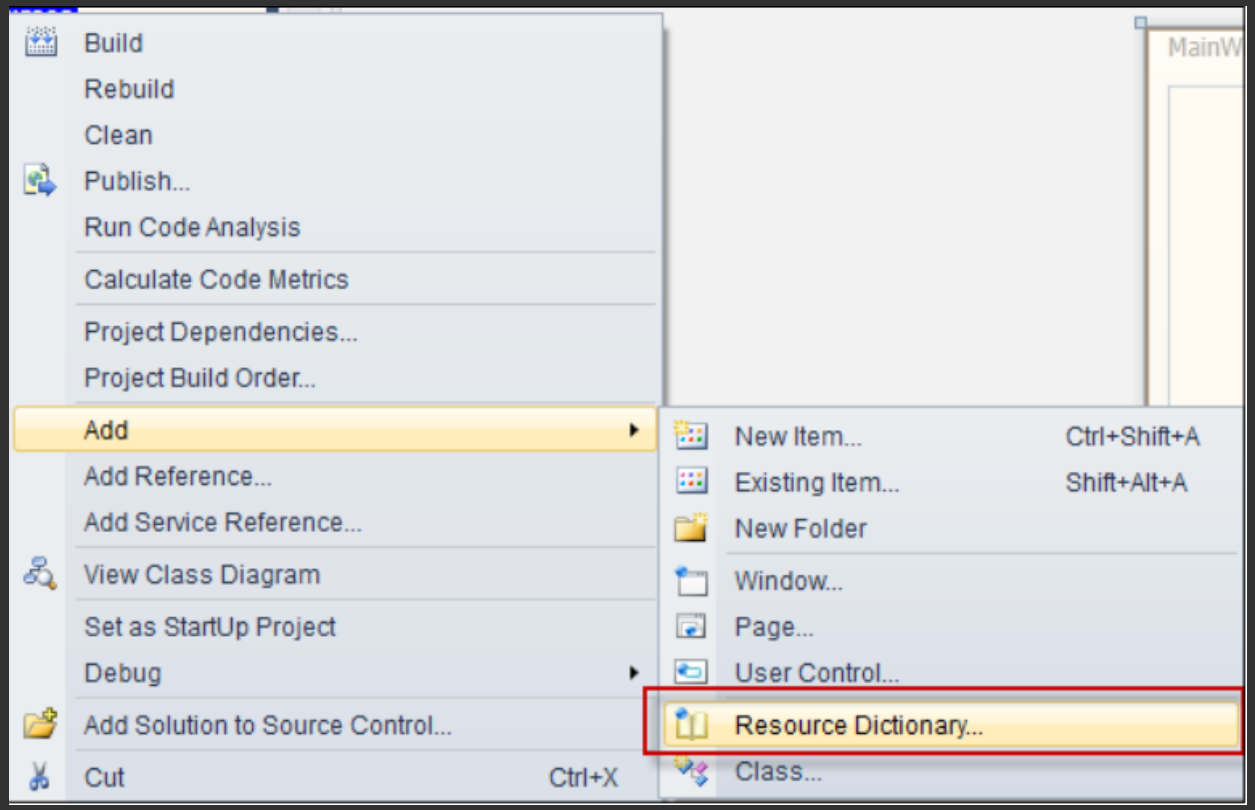
Доступ до бінарних ресурсів з іншої збірки

- WPF розпізнає pack URI на referenced assembly у формі:
  - /AssemblyReference;component/ResourceName
  - AssemblyReference може бути як простою назвою, так і включати версію (з префіксом "v") та/або public key token:
  - /MyAssembly;v2.0;4ac42a7f7bd64f34;component/images/apple.png
  - Це стандарт для full pack URI (prefxed by pack://application:,,,), and can also be an argument to Application.GetResourceStream
- Схема е працює з ресурсами, відміченими з Build Action рівним Content.
  - Обхідний шлях: використовувати full pack URI with a siteOfOrigin base.
  - Для попереднього прикладу потрібно змінити властивість Source:
  - Source="pack://siteOfOrigin:,,,/images/apple.png"

Управління логічними ресурсами

- Логічні ресурси: пензлі (brushes), геометрії (geometries), стилі, шаблони (templates) тощо.
  - Розміщення всіх таких ресурсів в одному файлі, наприклад, App.xaml, заважає maintainability.
  - Краще відокремити ресурси різних типів у власні файли.
  - Проте на них доведеться посилатись з одного спільного файлу, наприклад, App.xaml.

Управління логічними ресурсами



- ResourceDictionary може включати інші словники ресурсів за допомогою властивості MergedDictionaries.
  - ResourceDictionary може посилатись на довільну кількість словників ресурсів та мати власні ресурси.

- Властивість Source повинна вказувати на місце знаходження ResourceDictionary.
  - <ResourceDictionary Source="Resources/Brushes.xaml" />
- Аналогічна ідея при посиланні на логічні ресурси, які зберігаються в інших збірках (referenced assemblies).
  - Властивість Source має базуватись на синтаксисі pack URI.
  - Нехай Brushes.xaml розміщена в бібліотеці класів у папці Resources.
  - Основний додаток може включити її в інший ResourceDictionary:
  - <ResourceDictionary Source="/MyClassLibrary;component/Resources/Brushes.xaml" />

Дубльовані ключі

- Можлива проблема: кілька ресурсів з однаковими ключами з різних merged dictionaries.
  - Це не є помилкою та не викидає виключення.
  - Обирається об'єкт з останнього доданого словника ресурсів (which has a resource with that key).
  - Furthermore, if a resource in the current resource dictionary has the same key as the any of the resources in its merged dictionaries – it always wins out.

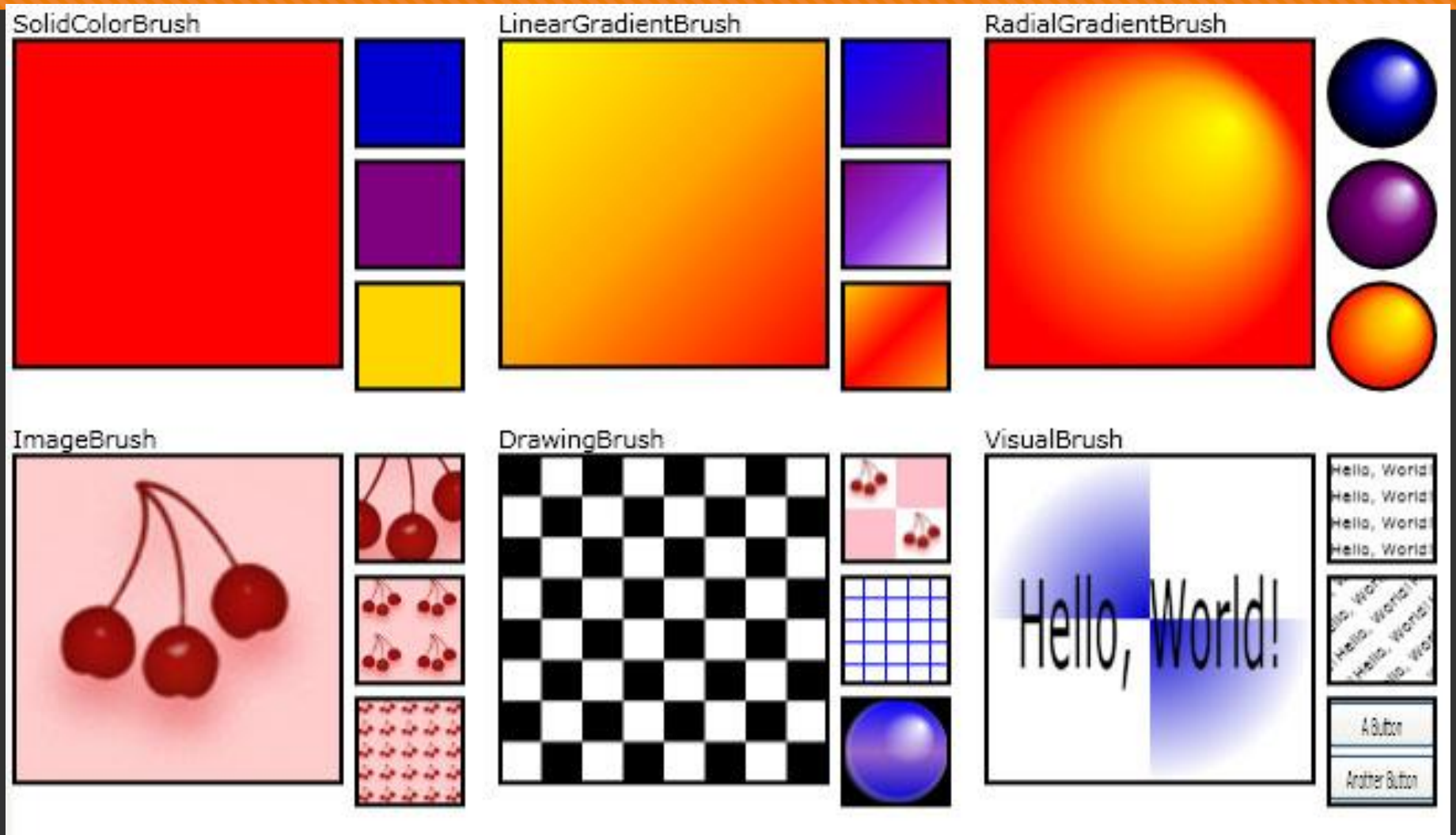
```
<ResourceDictionary>
   <SolidColorBrush Color="Blue" x:Key="brush1" />
   <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="Resources/Brushes2.xaml" />
      <ResourceDictionary Source="Resources/Brushes.xaml" />
   </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

Пензлі як логічний ресурс

- Для установки кольору використовуємо об'єкт класу System.Media.Brush
  - C#: **button1.Background=Brushes.Blue**
  - XAML: <Button Background="Blue" … />
- Більшість візуальних об'єктів дають можливість задати спосіб рисування.

Клас	Властивості
Border	BorderBrush, Background
Control	Background, Foreground
Panel	Background
Pen	Brush
Shape	Fill, Stroke
TextBlock	Background

Приклади пензлів

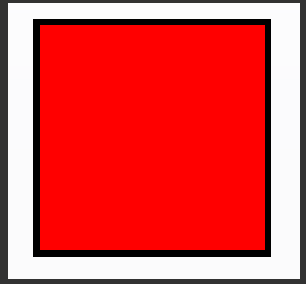


Заливка суцільним кольором

- Використовується об'єкт SolidColorBrush.
  - Наприклад, можна вказати ARGB-канали або використати один з кольорів, наданих у класі Colors.

```
// C#
Rectangle exampleRectangle = new Rectangle();
exampleRectangle.Width = 75;
exampleRectangle.Height = 75;
SolidColorBrush myBrush = new SolidColorBrush(Colors.Red);
exampleRectangle.Fill = myBrush;
```

```
<Rectangle Width="75" Height="75">
        <Rectangle.Fill>
                <SolidColorBrush Color="Red" />
        </Rectangle.Fill>
</Rectangle>
```



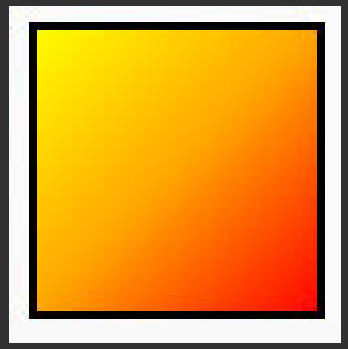
Рисування з лінійним градієнтом

- Використовується об’єкт LinearGradientBrush:
  - Использовании GradientStop объекты для указания цветов в градиенте и их положения.

```
// C#
Rectangle exampleRectangle = new Rectangle();
exampleRectangle.Width = 75;
exampleRectangle.Height = 75;
LinearGradientBrush myBrush = new LinearGradientBrush();
myBrush.GradientStops.Add(new GradientStop(Colors.Yellow, 0.0));
myBrush.GradientStops.Add(new GradientStop(Colors.Orange, 0.5));
myBrush.GradientStops.Add(new GradientStop(Colors.Red, 1.0));
exampleRectangle.Fill = myBrush;
```

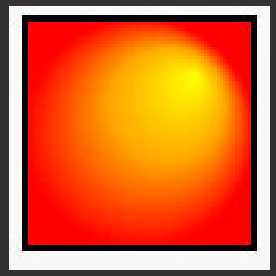
Рисування з лінійним градієнтом

```
<Rectangle Width="75" Height="75">
        <Rectangle.Fill>
                <LinearGradientBrush>
                        <GradientStop Color="Yellow" Offset="0.0" />
                        <GradientStop Color="Orange" Offset="0.5" />
                        <GradientStop Color="Red" Offset="1.0" />
                </LinearGradientBrush>
        </Rectangle.Fill>
</Rectangle>
```



Рисування з використанням радіального градієнта

- Використовується об'єкт RadialGradientBrush.
  - Радіальний градієнт поєднує кілька кольорів у крузі.
  - Як і з LinearGradientBrush, використовуються об'єкти класу GradientStop для вказування кольорів у градієнті.



```
Rectangle exampleRectangle = new Rectangle();
exampleRectangle.Width = 75;
exampleRectangle.Height = 75;
// Create a RadialGradientBrush and use it to paint the rectangle.
RadialGradientBrush myBrush = new RadialGradientBrush();
myBrush.GradientOrigin = new Point(0.75, 0.25);
myBrush.GradientStops.Add(new GradientStop(Colors.Yellow, 0.0));
myBrush.GradientStops.Add(new GradientStop(Colors.Orange, 0.5));
myBrush.GradientStops.Add(new GradientStop(Colors.Red, 1.0));
exampleRectangle.Fill = myBrush;
```

Рисування з використанням радіального градієнта

```
<Rectangle Width="75" Height="75">
        <Rectangle.Fill>
                <RadialGradientBrush GradientOrigin="0.75,0.25">
                        <GradientStop Color="Yellow" Offset="0.0" />
                        <GradientStop Color="Orange" Offset="0.5" />
                        <GradientStop Color="Red" Offset="1.0" />
                </RadialGradientBrush>
        </Rectangle.Fill>
</Rectangle>
```

- Додаткова інформація