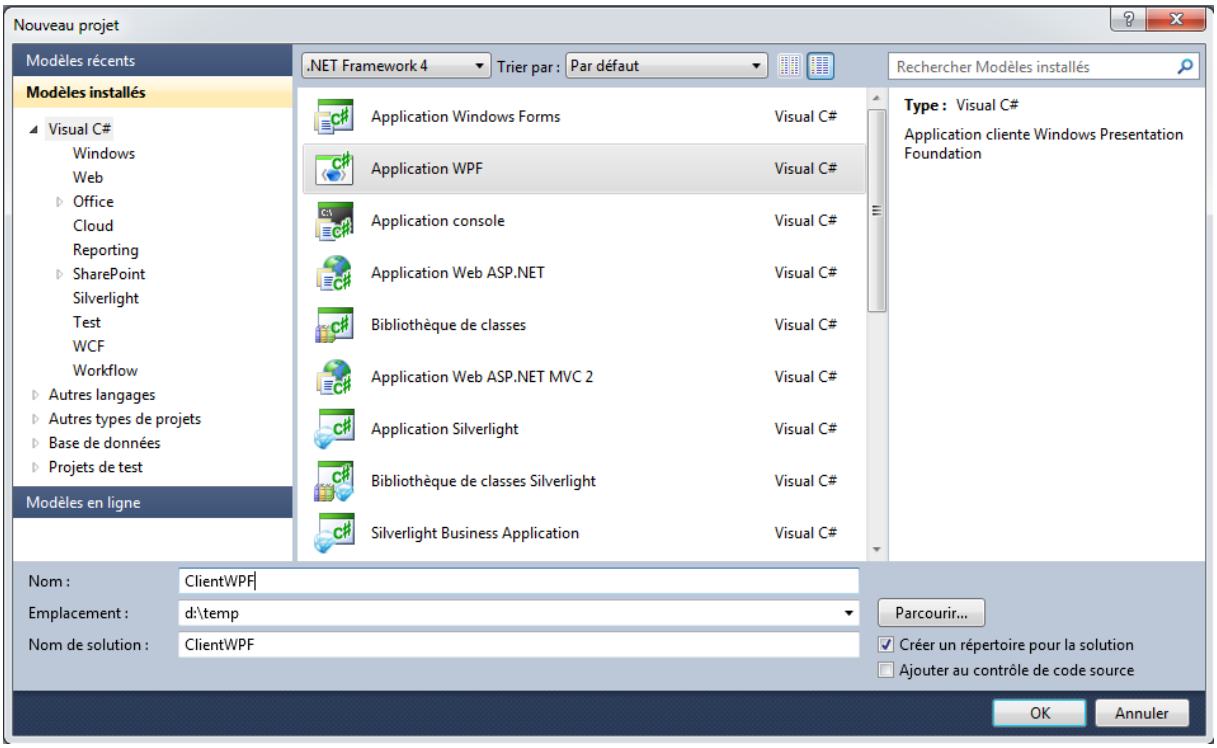
Enoncé complémentaire 5

Tips :

1. Création d'une ListBox contenant des éléments simples

1.1. Création d'un projet d'application WPF

On crée un nouveau projet Application WPF :

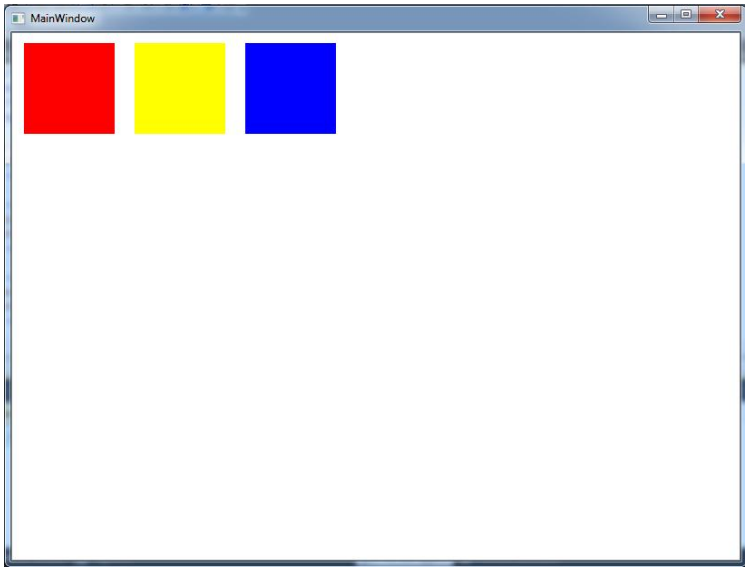


1.2. Création d'une ListBox dans la fenêtre principale

Nous allons ajouter une ListBox dans la fenêtre principale. Ce composant permet d'afficher une liste d'éléments. Nous allons définir l'agencement des éléments qui seront contenus dans notre liste en se basant sur le composant WrapPanel. Ceci permet de disposer les composants de notre liste de gauche à droite puis de haut en bas. Pour finir, nous ajoutons 3 rectangles de couleurs différentes dans notre liste. Voici le code XAML de notre fenêtre principale :

```
<Window x:Class="ClientWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="620" Width="820">
    <Grid>
        <ListBox x:Name="NotreListbox">
            <ListBox.ItemsPanel>
                <ItemsPanelTemplate>
                    <WrapPanel />
                </ItemsPanelTemplate>
            </ListBox.ItemsPanel>
            <Rectangle Height="100" Width="100" Margin="10">
                <Rectangle.Fill>
                    <SolidColorBrush Color="Red" />
                </Rectangle.Fill>
            </Rectangle>
            <Rectangle Height="100" Width="100" Margin="10">
                <Rectangle.Fill>
                    <SolidColorBrush Color="Yellow" />
                </Rectangle.Fill>
            </Rectangle>
            <Rectangle Height="100" Width="100" Margin="10">
                <Rectangle.Fill>
                    <SolidColorBrush Color="Blue" />
                </Rectangle.Fill>
            </Rectangle>
        </ListBox>
    </Grid>
</Window>
```

Les composants s'affichent donc de la façon suivante :



1.3. Création d'un style pour nos rectangle

Il est possible de définir un style pour nos différents rectangles. Nous allons mettre dans cet objet les différentes caractéristiques communes de nos rectangles comme « Height », « Width » et « Margin ». Le style s'appliquera ensuite à tous les rectangles qui y feront référence et il ne sera pas nécessaire de redéfinir ces caractéristiques pour chaque rectangle. On ajoute donc le style « NotreStyleDeRectangle » dans les ressources de notre fenêtre et on spécifie au rectangles d'utiliser cette ressource:

```
<Window x:Class="ClientWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="620" Width="820">
    <Window.Resources>
        <Style x:Key="NotreStyleDeRectangle" TargetType="Rectangle">
            <Setter Property="Width" Value="100"/>
            <Setter Property="Height" Value="100"/>
            <Setter Property="Margin" Value="10" />
        </Style>
    </Window.Resources>
    <Grid>
        <ListBox x:Name="NotreListbox">
            <ListBox.ItemsPanel>
                <ItemsPanelTemplate>
                    <WrapPanel />
                </ItemsPanelTemplate>
            </ListBox.ItemsPanel>
            <Rectangle Style="{StaticResource NotreStyleDeRectangle}">
                <Rectangle.Fill>
                    <SolidColorBrush Color="Red" />
                </Rectangle.Fill>
            </Rectangle>
            <Rectangle Style="{StaticResource NotreStyleDeRectangle}">
                <Rectangle.Fill>
                    <SolidColorBrush Color="Yellow" />
                </Rectangle.Fill>
            </Rectangle>
            <Rectangle Style="{StaticResource NotreStyleDeRectangle}">
                <Rectangle.Fill>
                    <SolidColorBrush Color="Blue" />
                </Rectangle.Fill>
            </Rectangle>
        </ListBox>
    </Grid>
</Window>
```

Les composants s'affichent de la même façon que précédemment.

1.4. Ajout d'une animation sur nos rectangles

On va ajouter deux animations sur notre rectangle grâce à l'objet DoubleAnimation que l'on inclue dans un objet Storyboard. La première animation sera déclenchée lors du passage de la souris sur le rectangle. La propriété Width du rectangle va alors augmenter avec une accélération 0.20 jusqu'à la valeur 300. Inversement, la deuxième animation va réduire la propriété du rectangle jusqu'à sa taille initiale. On modifie dans la ressource Style de la façon suivante :

```
<Style TargetType="Rectangle">
       <Setter Property="Width" Value="100"/>
       <Setter Property="Height" Value="100"/>
       <Setter Property="Margin" Value="10" />
       <Style.Triggers>
             <EventTrigger RoutedEvent="MouseEnter">
                   <BeginStoryboard>
                        <Storyboard>
                            <DoubleAnimation To="300" Duration="0:0:1"
                             AccelerationRatio="0.20" DecelerationRatio="0.20"
                             Storyboard.TargetProperty="Width"/>
                        </Storyboard>
                    </BeginStoryboard>
              </EventTrigger>
              <EventTrigger RoutedEvent="MouseLeave">
                   <BeginStoryboard>
                        <Storyboard>
                            <DoubleAnimation Duration="0:0:1"
                             AccelerationRatio="0.20" DecelerationRatio="0.20"
                             Storyboard.TargetProperty="Width"/>
                        </Storyboard>
                   </BeginStoryboard>
              </EventTrigger>
        </Style.Triggers>
</Style>
```

2. Utilisation d'une collection comme source de données d'une ListBox

Dans le cas précédent, nous avons défini en dur dans le code XAML les éléments contenus de notre ListBox. Nous souhaitons maintenant définir une collection de données qui sera liée à notre ListBox afin que celle-ci affiche les différents éléments de cette collection.

2.1. Création d'une collection de donnée

Nous créons pour cela une classe ImageObjet qui permet de contenir le contenu d'une image ainsi que son nom. Nous créons par ailleurs une classe ImageCollection qui hérite de la classe ObservableCollection et qui permet de contenir une collection d'ImageObjet. La classe ObservableCollection permet de stocker des données et peut d'être liée à une ListBox afin que la ListBox puisse en afficher son contenu :

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;

namespace ClientWPF
{
    public class ImageObjet
    {
        public String Nom { get; set; }
        public byte[] Image { get; set; }

        public ImageObjet(String Nom, byte[] Image)
        {
            this.Nom = Nom;
            this.Image = Image;
        }
    }

    public class ImageCollection : ObservableCollection<ImageObjet>
    {}
}
```

2.2. Affichage de notre collection

2.2.1. Création et liaison de la ressource ObjectDataProvider

Nous allons définir la ressource ObjectDataProvider :

```
<Window.Resources>
        <ObjectDataProvider x:Key="ImageCollection1"/>
</Window.Resources>
```

L'ObjectDataProvider permettra de faire le lien entre une ListBox et une collection de données. Nous allons pour cela créer une ImageCollection et la lier avec cet objet. Cette opération peut se faire à l'initialisation de notre fenêtre :

```
private ImageCollection imageCollection1;

public MainWindow()
{
      InitializeComponent();

      // On crée notre collection d'image et on y ajoute deux images
      imageCollection1 = new ImageCollection();
      imageCollection1.Add(new ImageObjet("celestial",
      lireFichier(@"d:\celestial.jpg")));
      imageCollection1.Add(new ImageObjet("pearlscale",
      lireFichier(@"d:\pearlscale.jpg")));

      // On lie la collectionau ObjectDataProvider déclaré dans le fichier XAML
      ObjectDataProvider imageSource =
      (ObjectDataProvider)FindResource("ImageCollection1");
      imageSource.ObjectInstance = imageCollection1;
}
```

2.2.2. Création du DataTemplate

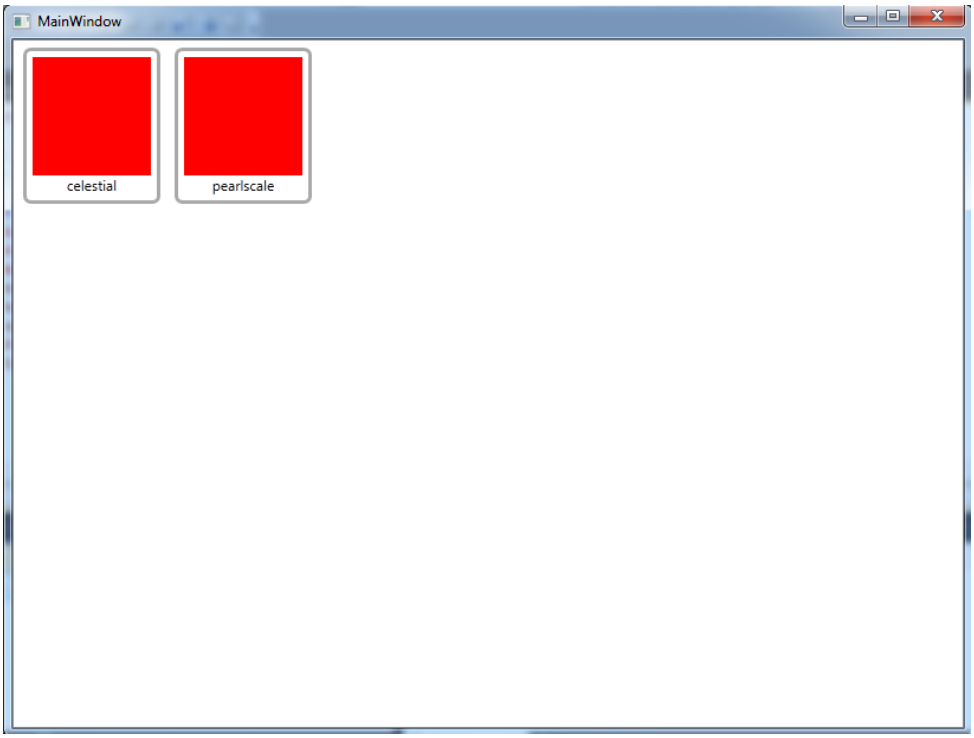
Nous allons maintenant rajouter dans nos ressources un objet DataTemplate. Cet objet permet de définir la façon dont chaque ImageObjet sera affiché dans la ListBox :

```
<Window.Resources>
        <ObjectDataProvider x:Key="ImageCollection1"/>
        <DataTemplate x:Key="ImageSourceTemplate">
            <Border Padding="5,5,5,5" Margin="5,5,5,5" BorderBrush="DarkGray"
              BorderThickness="3" CornerRadius="5">
                <StackPanel Orientation="Vertical">
                    <Rectangle  Width="100" Height="100">
                        <Rectangle.Fill>
                            <SolidColorBrush Color="Red"/>
                        </Rectangle.Fill>
                    </Rectangle>
                    <TextBlock Text="{Binding Path=Nom}"
              HorizontalAlignment="Center"/>
                </StackPanel>
            </Border>
        </DataTemplate>
</Window.Resources>
```

Pour la représentation de notre objet ImageObjet, nous avons défini une bordure qui contiendra un empilement d'un rectangle rouge et d'un TextBlock. Le TextBlock prendra pour valeur la propriété Nom de notre objet ImageObjet. Nous n'avons pas lié pour l'instant le contenu de notre image avec notre DataTemplate car il faut pour cela la convertir, ce que nous ferons dans la section 2.3. Nous devons maintenant modifier notre ListBox afin que celle-ci utilise les deux ressources précédemment créées (ObjectDataProvider et DataTemplate) :

```
<Grid>
        <ListBox x:Name="NotreListbox"
                 ItemsSource="{Binding}"
                 DataContext="{StaticResource ImageCollection1}"
                 ItemTemplate="{StaticResource ImageSourceTemplate}" >
            <ListBox.ItemsPanel>
                <ItemsPanelTemplate>
                    <WrapPanel />
                </ItemsPanelTemplate>
            </ListBox.ItemsPanel>
        </ListBox>
</Grid>
```

Lorsque l'on lance le projet, on s'aperçoit que notre ListBox affiche les deux éléments de notre collection :



Tous les éléments rajoutés dans la collection imageCollection1 seront donc affichés dans notre ListBox.

2.3. Implémentation d'un converter

Alors qu'il était facile de lier la propriété Nom de notre ImageObjet au contenu du TextBlock de notre DataTemplate, il est plus compliqué de lier notre fichier Image (en byte[]) à une BitmapImage. Nous devons pour cela créer une classe qui va se charger de la conversion. Tout d'abord, remplaçons le Rectangle de notre DataTemplate par une Image :

```
<Image  Width="100" Height="100" Stretch="Fill"  Source="{Binding Path=Image,
Converter={StaticResource ByteArrayToImageConverter}, Mode=Default}"/>
```

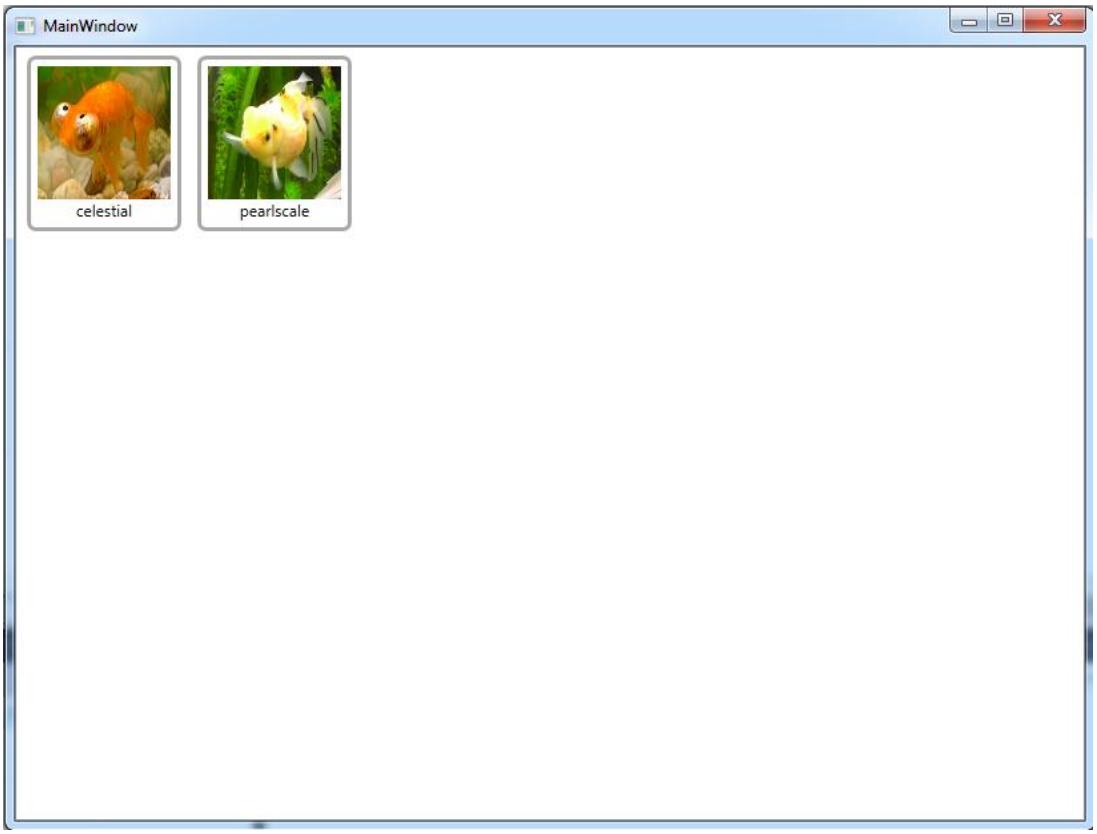
Ici, on indique que notre image contiendra la valeur de la propriété Image de notre ImageObjet, une fois celle-ci convertie en BitmapImage par la fonction ByteArrayToImageConverter. Voici l'implémentation de notre Converter :

```
using System;
using System.Windows.Data;
using System.Globalization;
using System.Windows.Media.Imaging;
using System.IO;

namespace WpfApplication1
{
        public class ByteArrayToImageConverter : IValueConverter
        {
            public object Convert(object value, Type targetType, object parameter,
              CultureInfo culture)
            {
                // crée un BitmapImage à partir d'un byte[]
                BitmapImage imageSource = null;
                byte[] array = (byte[])value;
                if (array != null)
                {
                    imageSource = new BitmapImage();
                    imageSource.BeginInit();
                    imageSource.StreamSource = new MemoryStream(array);
                    imageSource.EndInit();
                }
                return imageSource;
            }

            public object ConvertBack(object value, Type targetType, object
              parameter, CultureInfo culture)
            {
                throw new Exception("Non implementee");
            }
        }
}
```

Pour finir, il faut créer et lier la ressource ByteArrayToImageConverter utilisée dans la DataTemplate avec la classe que nous venons d'implémenter. Nous créons pour cela une référence « locale » sur le namespace ClientWPF, qui contient l'implémentation de notre converter. Nous créons alors notre ressource ByteArrayToImageConverter (précédemment utilisé dans notre DataTemplate) afin qu'elle pointe sur l'implémentation du converter :

```
<Window x:Class="ClientWPF.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:ClientWPF"
        Title="MainWindow" Height="620" Width="820">
    <Window.Resources>
        <local:ByteArrayToImageConverter x:Key="ByteArrayToImageConverter"/>
        <ObjectDataProvider x:Key="ImageCollection1"/>
        <DataTemplate x:Key="ImageSourceTemplate">
            <Border Padding="5,5,5,5" Margin="5,5,5,5" BorderBrush="DarkGray"
              BorderThickness="3" CornerRadius="5">
               <StackPanel Orientation="Vertical">
                   <Image  Width="100" Height="100" Stretch="Fill"
                    Source="{Binding Path=Image, Converter={StaticResource
                    ByteArrayToImageConverter}, Mode=Default}"/>
                   <TextBlock Text="{Binding Path=Nom}"
                    HorizontalAlignment="Center"/>
               </StackPanel>
            </Border>
        </DataTemplate>
    </Window.Resources>
```

Les images sont maintenant affichées dans les composants de notre ListBox :



3. Implémentation d'un Drag and Drop entre deux ListBox

3.1. Création des deux ListBox

Nous allons modifier notre Grid afin de contenir nos deux ListBox : pour cela nous allons définir 3 lignes dans notre Grid :

```
<Grid.RowDefinitions>
     <RowDefinition Height="*" />
     <RowDefinition Height="Auto" />
     <RowDefinition Height="*" />
</Grid.RowDefinitions>
```

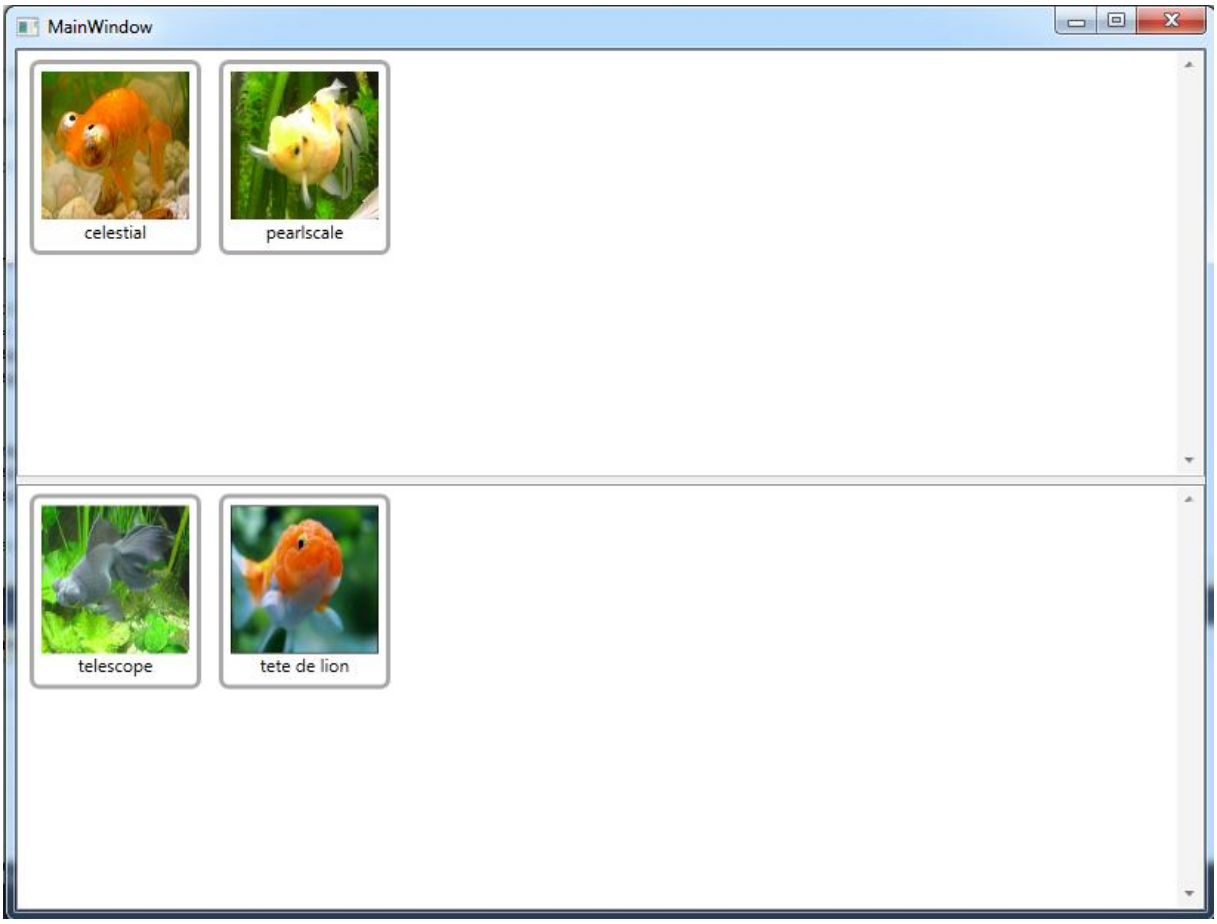
Les lignes 1 et 3 contiendront nos deux ListBox. La ligne 2 contiendra un GridSplitter qui permet de séparer visuellement nos deux ListBox. Ce séparateur sera mobile et permettra ainsi de laisser le choix à l'utilisateur de laisser plus de place à l'une ou l'autre des ListBox. On définit la Height à « Auto » pour la ligne 2 afin que le séparateur prenne toute la place dont il a besoin. On définit les Height des lignes 1 et 3 à « * » afin que les deux ListBox se partagent la place restante dans le composant mère. Comme nous allons avoir maintenant deux ListBox qui auront des caractéristiques communes, il est intéressant de créer un Style contenant ces caractéristiques communes, comme nous l'avions fait pour les rectangles dans la section 1.3 :

```
<Style x:Key="ImageListBoxStyle" TargetType="ListBox">
  <Setter Property="ItemsSource" Value="{Binding}" />
  <Setter Property="ItemTemplate" Value="{StaticResource ImageSourceTemplate}" />
  <Setter Property="ScrollViewer.VerticalScrollBarVisibility" Value="Visible" />
  <Setter Property="ScrollViewer.HorizontalScrollBarVisibility" Value="Disabled" />
  <Setter Property="ListBox.ItemsPanel">
        <Setter.Value>
              <ItemsPanelTemplate>
                    <WrapPanel />
              </ItemsPanelTemplate>
        </Setter.Value>
  </Setter>
</Style>
```

Nous avons aussi rajouté un Scroll horizontal dans notre Style afin que tous les éléments puissent y tenir quelle que soit la taille de la ListBox. Voici donc le contenu de notre Grid :

```
<Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <ListBox x:Name="ListBox1"
                 Style="{StaticResource ImageListBoxStyle}"
                 DataContext="{StaticResource ImageCollection1}"
                 Grid.Row="0"/>
        <GridSplitter HorizontalAlignment="Stretch"
                      ResizeBehavior="PreviousAndNext"
                      Height="5"
                      Grid.Row="1"/>
        <ListBox x:Name="ListBox2"
                 Style="{StaticResource ImageListBoxStyle}"
                 DataContext="{StaticResource ImageCollection2}"
                 Grid.Row="2"/>
</Grid>
```

Nous avons fait référence ici à un ObjectDataProvider « ImageCollection2 » pour la deuxième ListBox : il faut donc le déclarer en ressource et le lier à une nouvelle collection de type ImageCollection. Cette opération a été détaillée dans la section 2.2.1. On obtient le résultat suivant :



3.2. Implémentation du Drag and Drop

Nous allons maintenant ajouter le Drag and Drop entre nos deux ListBox. Les étapes suivantes, détaillant l'implémentation de cette fonctionnalité, sont extraites du site web http://marlongrech.wordpress.com/2007/12/28/drag-drop-using-listboxes-part-1/

Nous devons tout d'abord lier deux fonctions à deux événements :

- La première, « ImageDragEvent », sera appelée lors de l'événement « PreviewMouseLeftButtonDown », qui correspond au début de notre Drag and Drop.
- La seconde, « ImageDropEvent », sera appelée lors de l'événement « Drop », qui correspond à la fin de notre Drag and Drop

Voici le code modifié :

```
<ListBox x:Name="ListBox1"
                 Style="{StaticResource ImageListBoxStyle}"
                 DataContext="{StaticResource ImageCollection1}"
                 PreviewMouseLeftButtonDown="ImageDragEvent"
                 Drop="ImageDropEvent"
                 Grid.Row="0"/>
<GridSplitter HorizontalAlignment="Stretch"
                      ResizeBehavior="PreviousAndNext"
                      Height="5"
                      Grid.Row="1"/>
<ListBox x:Name="ListBox2"
                 Style="{StaticResource ImageListBoxStyle}"
                 DataContext="{StaticResource ImageCollection2}"
                 PreviewMouseLeftButtonDown="ImageDragEvent"
                 Drop="ImageDropEvent"
                 Grid.Row="2"/>
```

Ici, on a choisi les mêmes fonctions de traitement du Drag and Drop pour les deux listes mais il est possible d'utiliser des fonctions différentes si l'on souhaite que ces événements provoquent un traitement différent pour chaque ListBox. On assigne aussi à « True » la propriété « AllowDrop » dans notre style « ImageListBoxStyle » :

```
<Setter Property="AllowDrop" Value="True" />
```

On doit donc maintenant implémenter les fonctions ImageDragEvent et ImageDropEvent. Pour cela, on ajoute le code suivant au code-behind de notre Fenêtre (MainWindow.xaml.cs) :

```
ListBox dragSource = null;

// On initie le Drag and Drop
private void ImageDragEvent(object sender, MouseButtonEventArgs e)
{
      ListBox parent = (ListBox)sender;
      dragSource = parent;
      object data = GetDataFromListBox(dragSource, e.GetPosition(parent));
      if (data != null)
      {
          DragDrop.DoDragDrop(parent, data, DragDropEffects.Move);
      }
}

// On ajoute l'objet dans la nouvelle ListBox et on le supprime de l'ancienne
private void ImageDropEvent(object sender, DragEventArgs e)
{
      ListBox parent = (ListBox)sender;
      ImageObjet data = (ImageObjet)e.Data.GetData(typeof(ImageObjet));
      ((IList)dragSource.ItemsSource).Remove(data);
      ((IList)parent.ItemsSource).Add(data);
}

// On récupére l'objet que que l'on a dropé
private static object GetDataFromListBox(ListBox source, Point point)
{
      UIElement element = (UIElement)source.InputHitTest(point);
      if (element != null)
      {
           object data = DependencyProperty.UnsetValue;
           while (data == DependencyProperty.UnsetValue)
                {
                    data =
                     source.ItemContainerGenerator.ItemFromContainer(element);
                    if (data == DependencyProperty.UnsetValue)
                    {
                        element = (UIElement)VisualTreeHelper.GetParent(element);
                    }
                    if (element == source)
                    {
                        return null;
                    }
                }
            if (data != DependencyProperty.UnsetValue)
            {
                    return data;
             }
        }
         return null;
}
```

Des explications plus détaillées sur l'implémentation de ces fonctions sont disponibles sur le site http://marlongrech.wordpress.com/2007/12/28/drag-drop-using-listboxes-part-1/

Il est maintenant possible d'effectuer des Drag and Drop entre nos deux ListBox.