# Extending Views and Adding Custom Attributes in Your Xamarin.Android App


Lori Lalonde
Senior Consultant
@loriblalonde
solola.ca


EVOLVE16

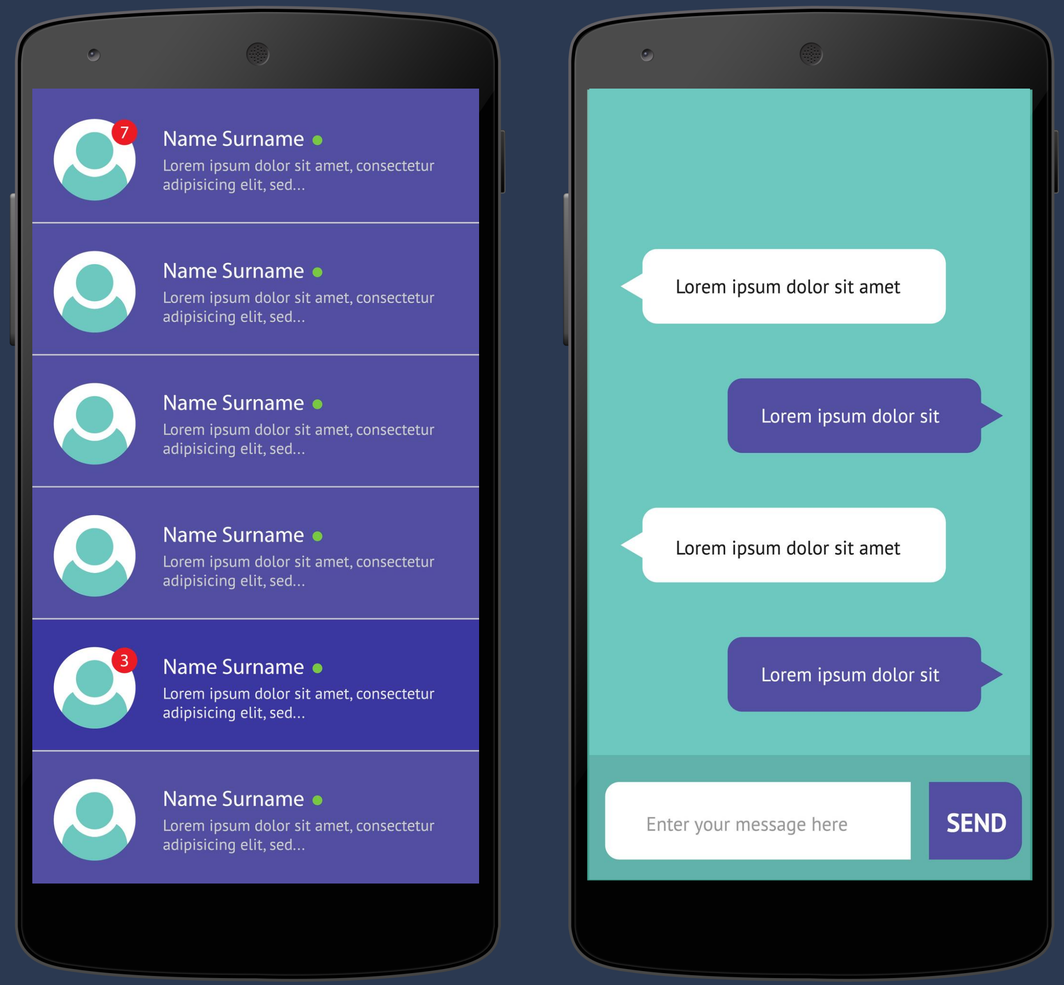
Xamarin MOST VALUABLE PROFESSIONAL

Microsoft® MVP Most Valuable Professional

western devs

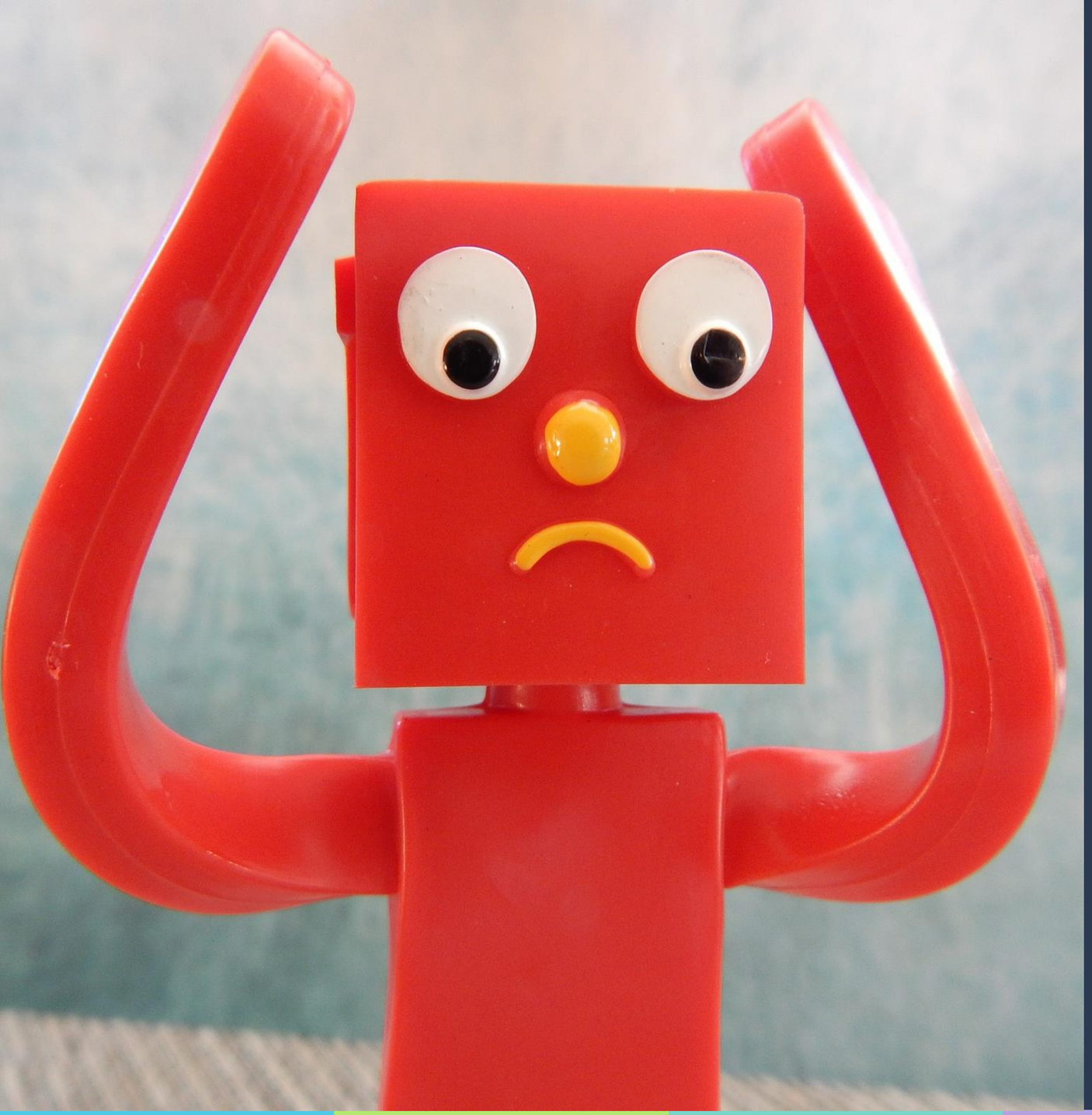
# Why?

- App design is not restricted to controls available on the platform
- Visual elements often need to be customized to achieve the desired look and feel



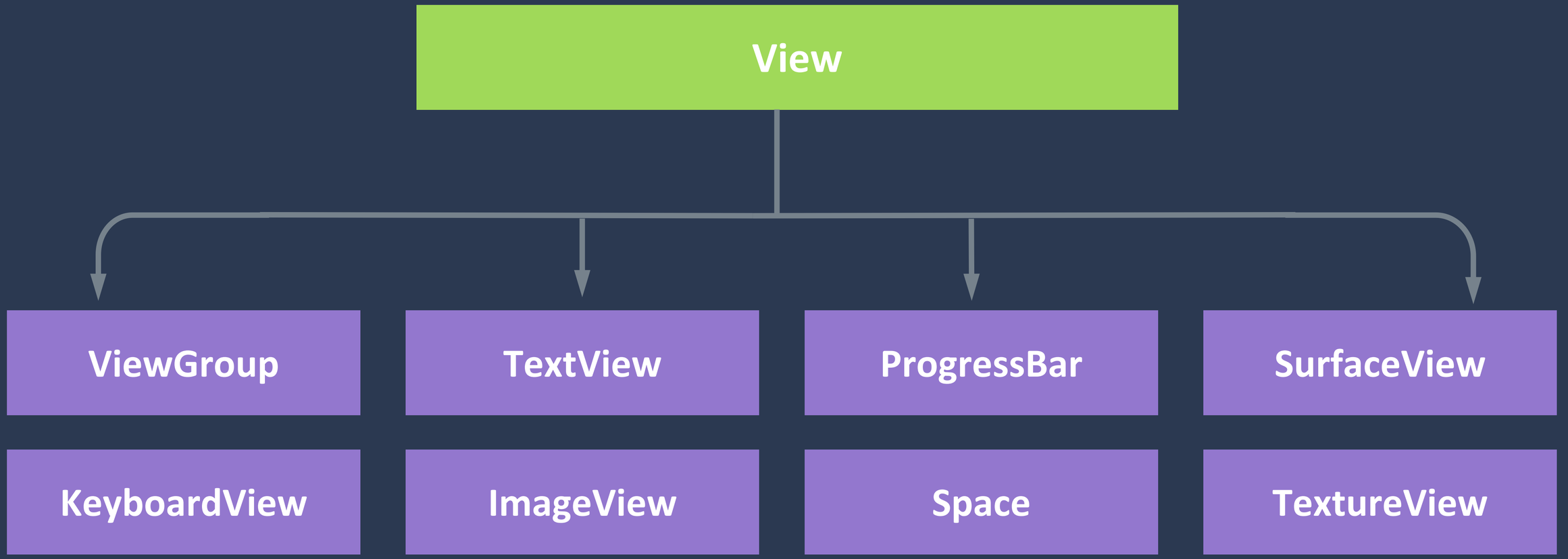
# Advantages

- ✓ Encapsulate custom behavior
- ✓ Create reusable component for use in multiple views or apps

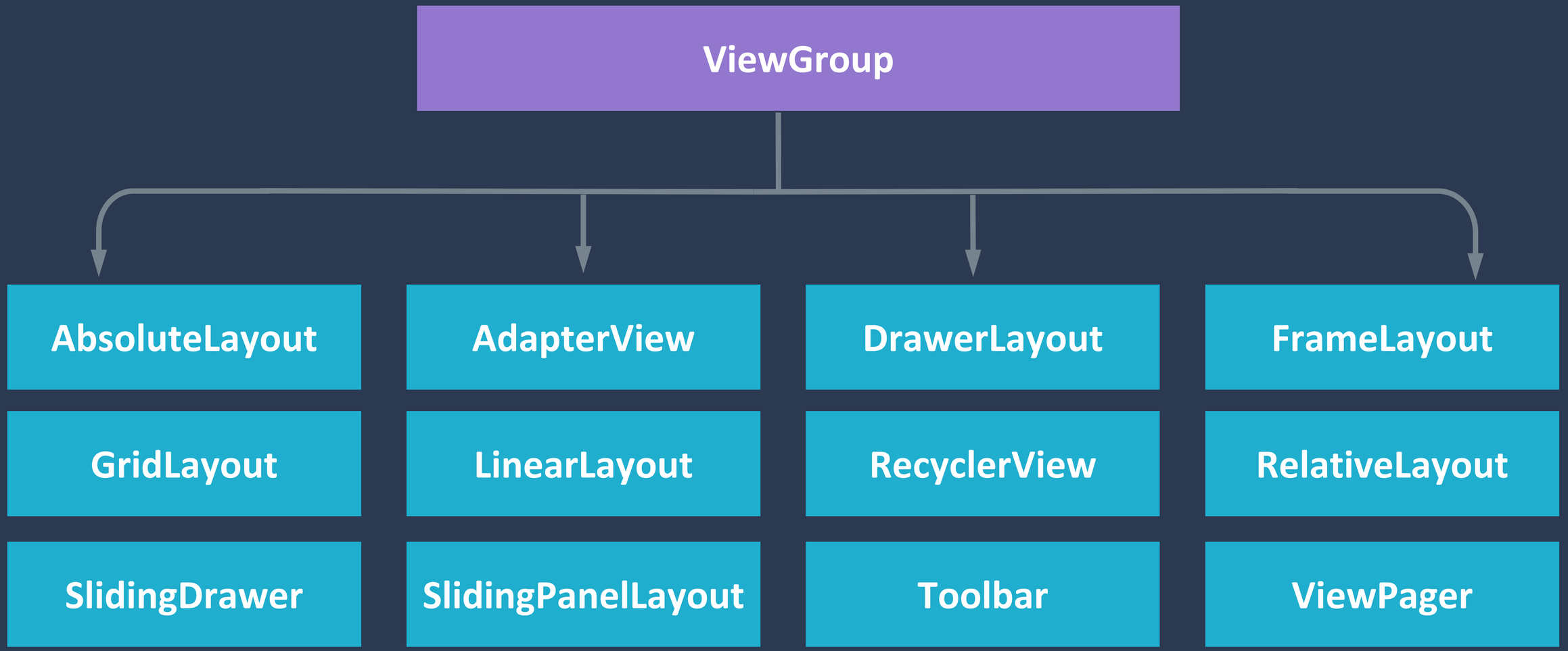


# Where do I start?

# It All Begins With a View



# ViewGroup Subclasses



# Subclass an Existing View

At a minimum, include a constructor that takes a Context and IAttributeSet

```
public class MyBalloonView : View
{
	public MyBalloonView (Context context, IAttributeSet attrs) : base(context, attrs)
	{
	}
}
```

# Define Custom Properties

- Determine the customizations needed for your view
- Expose properties to enable those customizations

```
public Color BallonColor { get; set; }

public Color AccentColor { get; set; }

public bool ShowEmojis { get; set; }

public float TextSize { get; set; }

public float TriangleHeight { get; set; }
```

# Method Overrides

| | |
|---|---|
| OnFinishInflate () | OnTrackballEvent (MotionEvent) |
| OnMeasure (int, int) | OnTouchEvent (MotionEvent) |
| OnLayout (boolean, int, int, int, int) | OnFocusChanged (boolean, int, Rect) |
| OnSizeChanged (int, int, int, int) | OnWindowFocusChanged (boolean) |
| OnDraw (Canvas) | OnAttachedToWindow () |
| OnKeyDown (int, KeyEvent) | OnDetachedFromWindow () |
| OnKeyUp (int, KeyEvent) | OnWindowVisibilityChanged (int) |

# Add the Custom View to a Layout

Reference the custom view using its fully qualified name:

[package name].[class name]

```
<customballoon.MyBalloonView
  android:layout_width=" wrap_content"
  android:layout_height="wrap_content" />
```

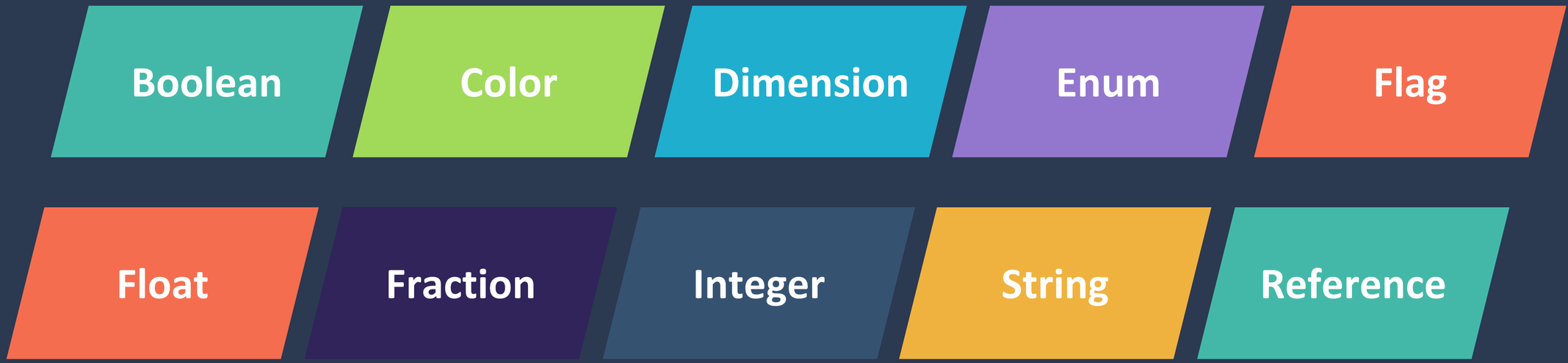
# Demo: Extending the View

# Define Custom Attributes

Enables design-time configuration of custom view in the XML layout

```
<declare-styleable name=“MyBalloonView">
    <attr name="showEmoji" format="boolean" />
    <attr name="balloonColor" format="color" />
    <attr name="text" format=“string" />
    <attr name="textSize" format=“dimension" />
    <attr name="triangleHeight" format= "integer" />
</declare-styleable>
```

# Attribute Types



# Retrieve Attribute Values

IAttributeSet contains the collection of attribute values from the view

```
public MyBalloonView(Context context, IAttributeSet attrs)
    : base(context, attrs)
{
        TypedArray typedArray =
            context.Theme.ObtainStyledAttributes(
                attrs, Resource.Styleable.MyBalloonView, 0, 0);

        this.Text = typedArray.GetString(
            Resource.Styleable.MyBalloonView_text);

        typedArray.Recycle();
}
```

# Retrieve Attribute Values

To access the attribute values, pass the attribute set to the context's ObtainStyledAttributes method

The resulting collection of attribute values are represented in a TypedArray

```
public MyBalloonView(Context context, IAttributeSet attrs)
    : base(context, attrs)
{
    TypedArray typedArray =
        context.Theme.ObtainStyledAttributes(
            attrs, Resource.Styleable.MyBalloonView, 0, 0);

    this.Text = typedArray.GetString(
        Resource.Styleable.MyBalloonView_text);

    typedArray.Recycle();
}
```

# Retrieve Attribute Values

Retrieve values for each attribute using TypedArray's Get methods:

- GetBoolean
- GetColor
- GetDimension
- GetString

...and more

```
public MyBalloonView(Context context, IAttributeSet attrs)
    : base(context, attrs)
{
    TypedArray typedArray =
        context.Theme.ObtainStyledAttributes(
            attrs, Resource.Styleable.MyBalloonView, 0, 0);

    this.Text = typedArray.GetString(
        Resource.Styleable.MyBalloonView_text);

    typedArray.Recycle();
}
```

# Retrieve Attribute Values

Release TypedArray from allocated memory by calling Recycle()

```
public MyBalloonView(Context context, IAttributeSet attrs)
    : base(context, attrs)
{
        TypedArray typedArray =
            context.Theme.ObtainStyledAttributes(
                attrs, Resource.Styleable.MyBalloonView, 0, 0);

        this.Text = typedArray.GetString(
            Resource.Styleable.MyBalloonView_text);

        typedArray.Recycle();
}
```

# Invalidate the Layout

Force the View to redraw when attribute values have changed

```
private Color _balloonColor;
public Color BalloonColor
{
    get { return _balloonColor; }
    set
      {
        _balloonColor = value;
        Invalidate();
        RequestLayout();
    }
}
```

# Configure Custom Attributes in the Layout

Define an alias for the custom attributes' namespace:

```
xmlns:custom="http://schemas.android.com/apk/res/[your package name]"
```

# Configure Custom Attributes in the Layout

Define an alias for the custom attributes' namespace:

```
xmlns:custom="http://schemas.android.com/apk/res/[your package name]"
```

Or, if contained in a separate library:

```
xmlns:custom="http://schemas.android.com/apk/res-auto"
```

# Configure Custom Attributes in the Layout

Once custom attributes are defined, you may set their respective values in the XML layout

```
<customballoon.MyBalloonView
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    custom:textSize="18dp"
    custom:textColor=“@color/White"
    custom:showEmoji="true"
    custom:triangleHeight=“20"
    custom:balloonColor=“@color/Purple"
    custom:accentColor=“@color/Pink" />
```

# Demo: Using Custom Attributes

# Questions?

# Thanks!


Lori Lalonde
Senior Consultant
@loriblalonde


solola.ca

westerndevs.com

EVOLVE16

Xamarin MOST VALUABLE PROFESSIONAL

Microsoft® MVP Most Valuable Professional

western devs