Objectives

1. Populate a `ListView` using an `ArrayAdapter`

2. Handle list-item click events

3. Implement a custom adapter

4. Use layout recycling and the view-holder pattern

5. Enable fast scrolling and code a section indexer
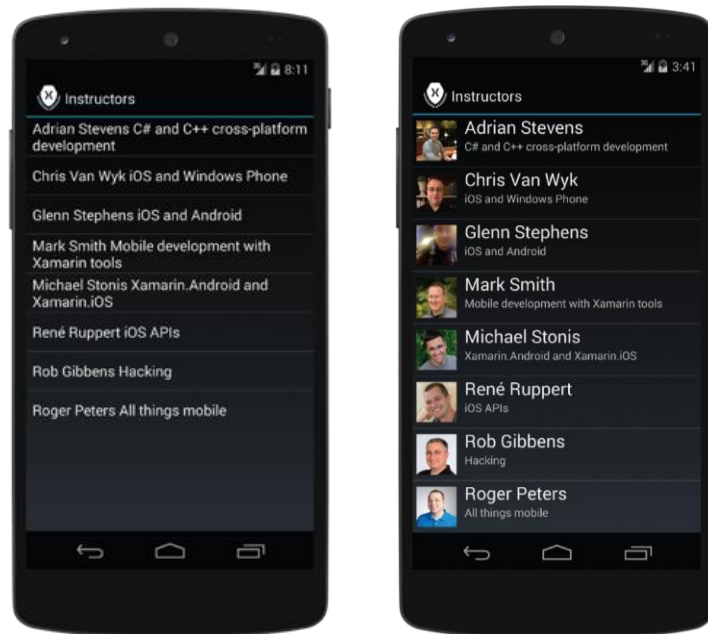
Populate a ListView using an ArrayAdapter

Tasks

1. Add a **ListView** to a UI

2. Use **ArrayAdapter** to populate a **ListView**

3. See the limitations of **ArrayAdapter**

What is a ListView?

❖ **`ListView`** displays a data collection as a sequence of visual rows
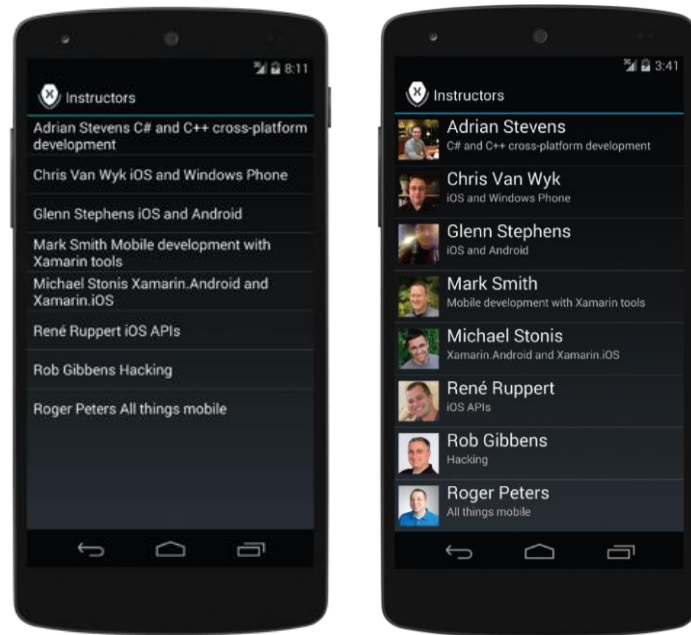
Rows can be simple strings or complex layouts with many views

Create a ListView

❖ Android List Views are created by declaring a **ListView** in a layout

```
<LinearLayout … >
    <ListView
        android:id="@+id/coursesListView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <TextView android:id="@android:id/empty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="No Courses found" />
</LinearLayout>
```

Define a **ListView** and assign an **id**

Optionally create a **TextView** with an id of empty to display default content

There is also an **Activity** subclass named **ListActivity** that presents a full screen **ListView**

What is an Adapter?

❖ An **Adapter** is a class that creates and populates the rows in a
   `ListView`

```
var l = new List<Instructor>();
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
```
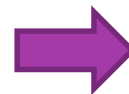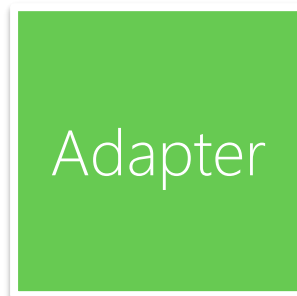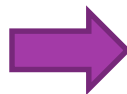
Adapter

This Adapter creates
each row with an image
and two pieces of text

What is an ArrayAdapter?

❖ **ArrayAdapter** is a built-in adapter that populates a row using only a single string from your data

```csharp
var l = new List<Instructor>();
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
l.Add(new Instructor() { ... });
```

Array Adapter

Calls **ToString** on the **Instructor** and uses it to populate a **TextView**
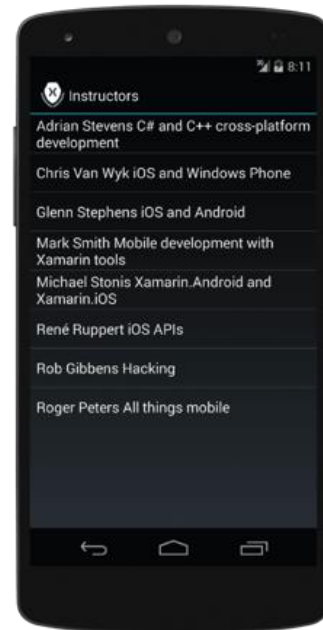
How to Use ArrayAdapter

❖ **ArrayAdapter** needs a layout file with a **TextView** and a data collection

```
var data = new List<Instructor>();
...

var adapter = new ArrayAdapter<Instructor>(this, layoutFileId, data);

var list = FindViewById<ListView>(Resource.Id.myList);
list.Adapter = adapter;
```

Id of the layout file to use for each row

Collection to display

Flash Quiz

Flash Quiz

① How are the rows in a **ListView** created?

   a) The **ListView** creates them using a Data Template

   b) The **ListView** asks the Adapter for each row as needed

   c) Rows are always strings so there is no need to create them

Flash Quiz

① How are the rows in a `ListView` created?

    a) The `ListView` creates them using a Data Template

    b) <u>The **`ListView`** asks the Adapter for each row as needed</u>

    c) Rows are always strings so there is no need to create them

Flash Quiz

② What is the key limitation of the standard `ArrayAdapter` implementation?

    a) Data objects must be in an array

    b) The rows it builds do not support `ItemClick` events

    c) It can only populate one `TextView`

Flash Quiz

② What is the key limitation of the standard `ArrayAdapter` implementation?

   a) Data objects must be in an array

   b) The rows it builds do not support `ItemClick` events

   c) <u>It can only populate one `TextView`</u>

Flash Quiz

③ How does **ArrayAdapter** convert the code-behind data into a string?

    a) Calls `ToString`

    b) Serializes the object to XML

    c) Uses reflection to get the first string property in the object

Flash Quiz

③ How does **ArrayAdapter** convert the code-behind data into a string?

    a) <u>Calls **ToString**</u>

    b) Serializes the object to XML

    c) Uses reflection to get the first string property in the object

Summary

1. Add a **ListView** to a UI

2. Use **ArrayAdapter** to populate a **ListView**

3. See the limitations of **ArrayAdapter**

Handle list-item click events

Tasks

1. Subscribe to the `ListView.ItemClick` event
2. Determine which list items was clicked
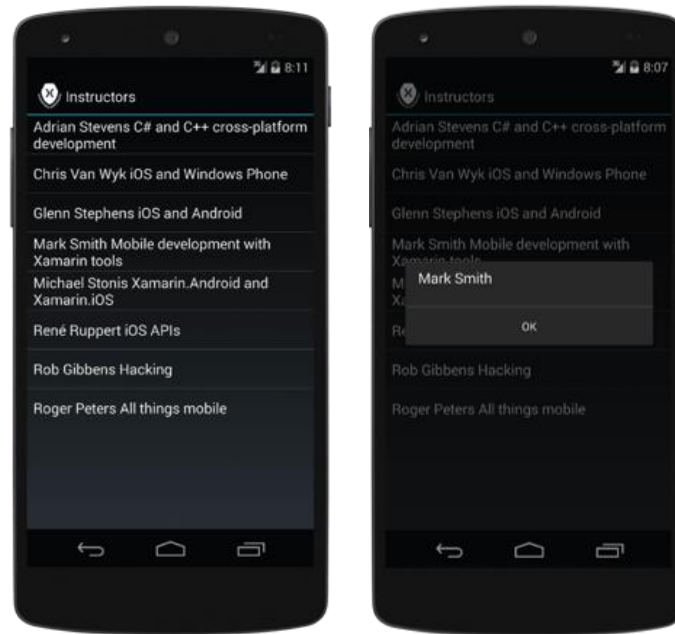
How to Handle ItemClick

❖ Subscribe to **ListView.ItemClick** to respond to user clicks

```csharp
var l = FindViewById<ListView>(Resource.Id.myList);

l.ItemClick += OnItemClick;
```

```csharp
void OnItemClick(object sender, AdapterView.ItemClickEventArgs e)
{
    var position = e.Position;
    ...
}
```

Event args contain the position of the clicked item

Individual Exercise

Handle list-item click events

Xamarin University

Summary

1. Subscribe to the
   `ListView.ItemClick` event
2. Determine which list items was
   clicked
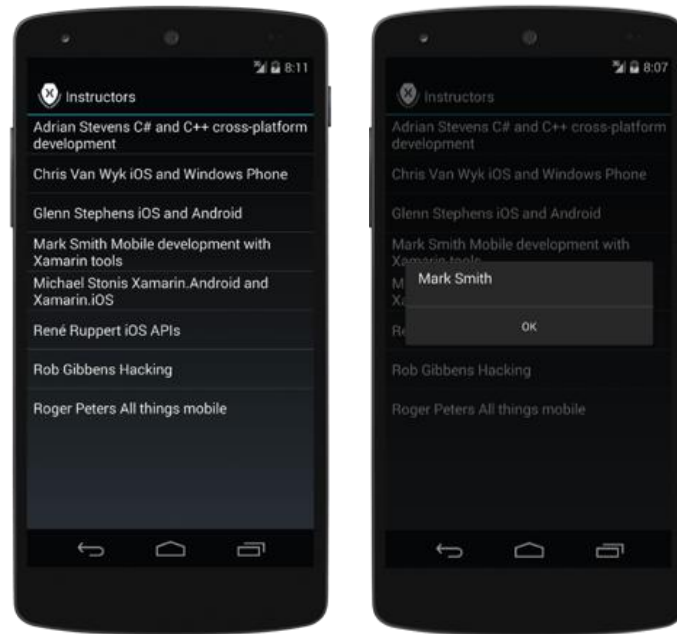
Implement a custom adapter

Tasks

1. Why build a custom adapter?

2. Inflate a layout file with `LayoutInflater`

3. Code a custom Adapter

Why build a custom adapter?

❖ **ArrayAdapter** is fine for small views of data, but inefficient when dealing with more than a page or two because it's built around Java objects

C# Wrapper

Java ArrayList

Java Object

Java Object

Java Object

Java Object

C# object

Android's **ArrayList** only understands Java object, so all your C# things get boxed" into Java types

Why build a custom adapter?

❖ `ArrayAdapter` is fine for small views of data, but inefficient when dealing with more than a page or two because it's built around Java objects

C# Wrapper

Java ArrayList

Java Object

Java Object

Java Object

Java Object

C# object

Objects are not collected as quickly because the collections must be synchronized between Mono and Android

Mono Garbage Collector

Android Garbage Collector
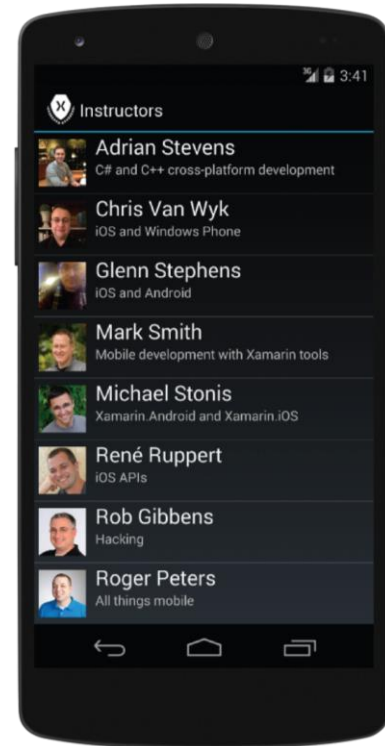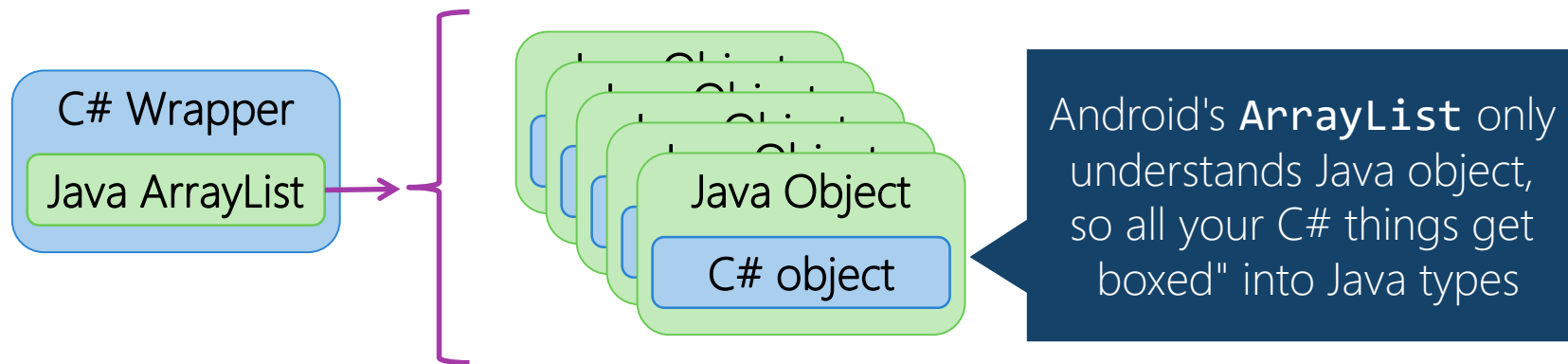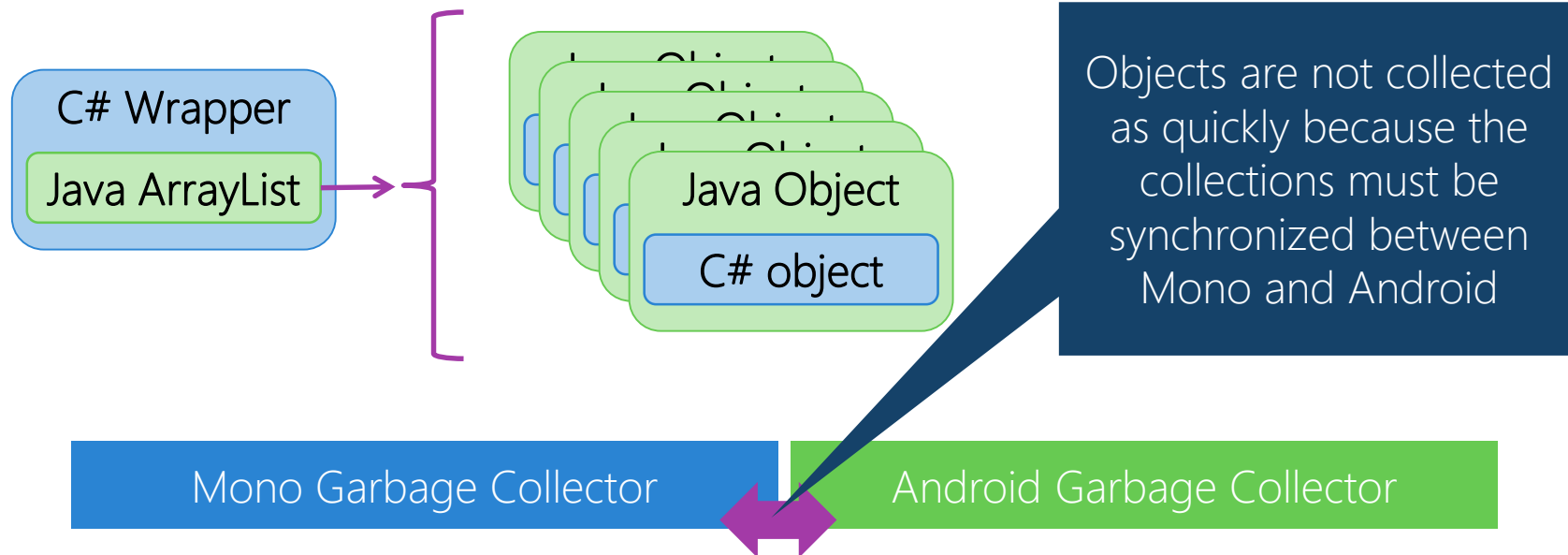
Why build a custom adapter?

❖ `ArrayAdapter` can only generate text-based rows for the `ListView`

I get this:
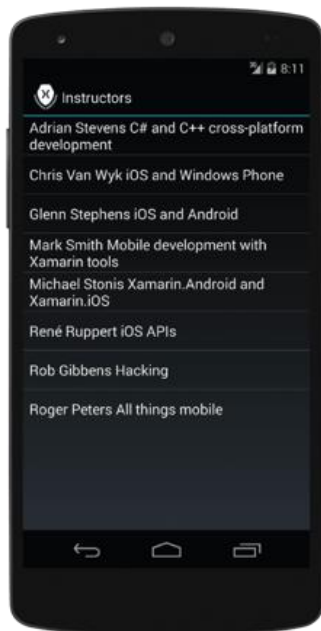
But want this:
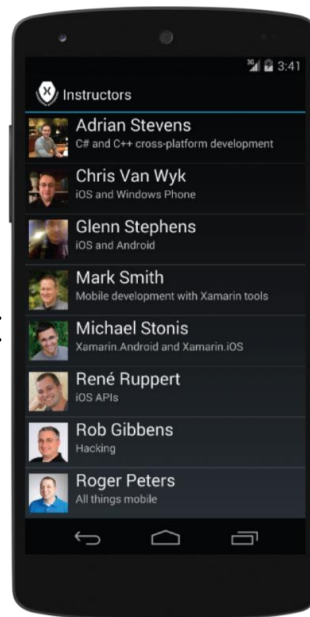
What is Inflation?

❖ *Inflation* is the process of instantiating the contents of a layout file

```
<RelativeLayout ... >
  <ImageView ... />
  <TextView  ... />
  <TextView  ... />
</RelativeLayout ... >
```

Inflation creates a view hierarchy from a layout file

```
RelativeLayout
        │
        ├──── ImageView
        │
        ├──── TextView
        │
        └──── TextView
```
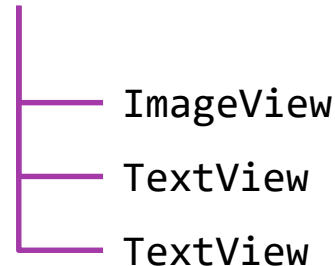
What is a LayoutInflater?

❖ Library class **LayoutInflater** performs inflation

```
<RelativeLayout ... >
  <ImageView ... />
  <TextView  ... />
  <TextView  ... />
</RelativeLayout ... >
```

Layout
Inflater

RelativeLayout

```
RelativeLayout
        ├──── ImageView
        ├──── TextView
        └──── TextView
```

LayoutInflater takes a resource file id and returns a **View** hierarchy
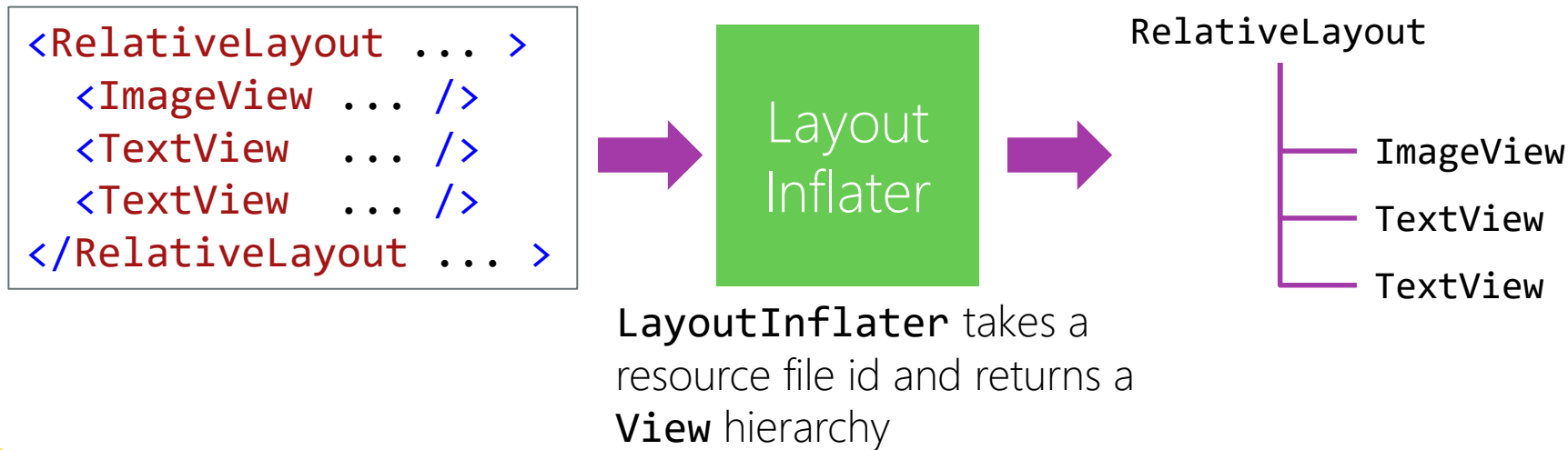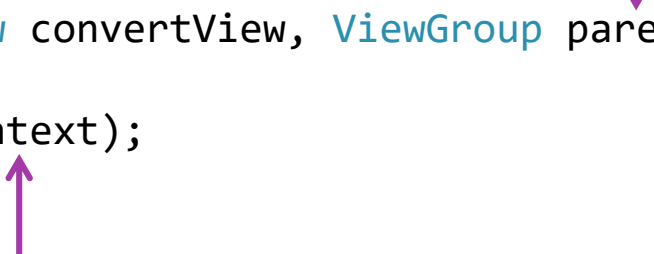
Android uses the spelling *inflater* rather than *inflator* and we will follow that convention.

Inflater access

❖ Your adapter needs an *inflater,* it is typical to <u>use the parent view</u> passed to your adapter's **GetView** method

The **ViewGroup** that will contain your inflated layout

```
public override View GetView(int position, View convertView, ViewGroup parent)
{
    var inflater = LayoutInflater.From(parent.Context);
    ...
}
```

Android allows you to get a **LayoutInflater** from a **Context**

What is BaseAdapter<T>?

❖ **BaseAdapter<T>** is a base class for custom adapters, it declares the four methods every Adapter must provide

```
public abstract class BaseAdapter<T> : BaseAdapter
{ ...
  public abstract View GetView(int position, View convertView, ViewGroup parent);

  public abstract T this[int position] { get; }
  public abstract int  Count { get; }
  public abstract long GetItemId(int position);
}
```

Generate a row   Information about the collection

What is IListAdapter?

❖ **ListView** expects an **IListAdapter** (which extends **IAdapter**) to turn data into Android views

❖ This is what **ArrayAdapter** and **BaseAdapter<T>** implement

```
class MyAdapter :
    TrainerDb, IListAdapter
{

  public View GetView(...) {
   ...
  }
  ...
}
```

Can implement this directly to combine data provider and adapter into a single class

How to code GetView

❖ `GetView` produces a row by inflating a layout file and populating the views with code-behind data

```
<RelativeLayout ... >
  <ImageView ... />
  <TextView  ... />
  <TextView  ... />
</RelativeLayout ... >
```

```
Name     : Adrian Stevens
Specialty: "C# and C++ ... "
ImageUrl : images/adrian.jpg
Biography: " ... "
```

Adapter

Adrian Stevens
C# and C++ cross-platform development
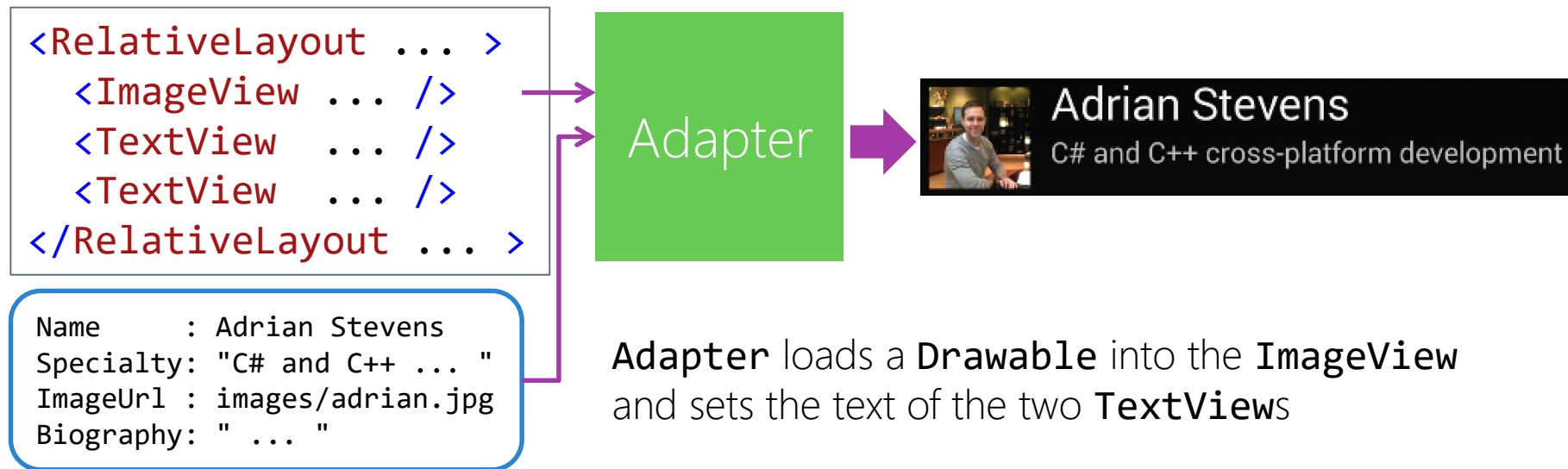
Adapter loads a **Drawable** into the **ImageView** and sets the text of the two **TextView**s

Individual Exercise

Implement a custom adapter

Xamarin University

Summary

1. Why build a custom adapter?
2. Inflate a layout file with `LayoutInflater`
3. Code a custom Adapter
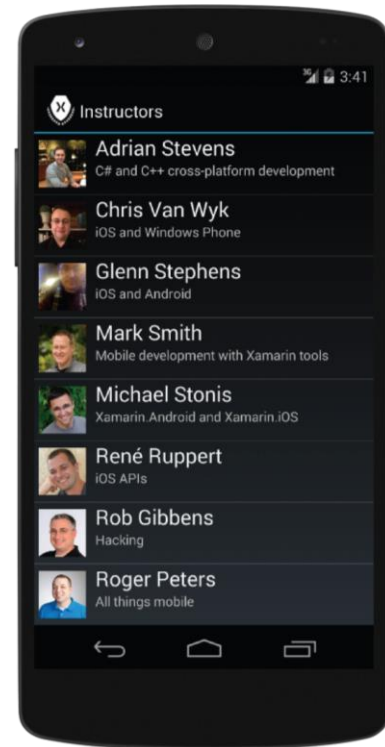
Use layout recycling and the view-holder pattern

Tasks

1. Reuse inflated layouts to reduce memory usage

2. Cache view references to increase performance
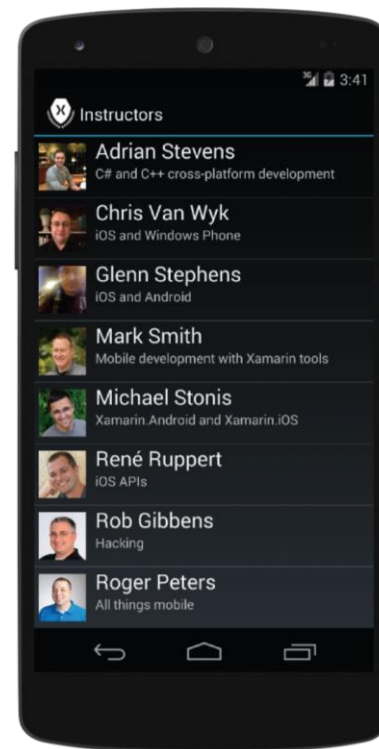
ListView Layout Reuse

❖ **ListView** maintains populated layouts only for rows that are visible to the user, non-visible layouts are recycled

As user scrolls down, the top layout is no longer needed, it is passed to **GetView** to be refilled with new data and added at bottom

Instructors

Adrian Stevens
C# and C++ cross-platform development

Chris Van Wyk
iOS and Windows Phone

Glenn Stephens
iOS and Android

Mark Smith
Mobile development with Xamarin tools

Michael Stonis
Xamarin.Android and Xamarin.iOS

René Ruppert
iOS APIs

Rob Gibbens
Hacking

Roger Peters
All things mobile
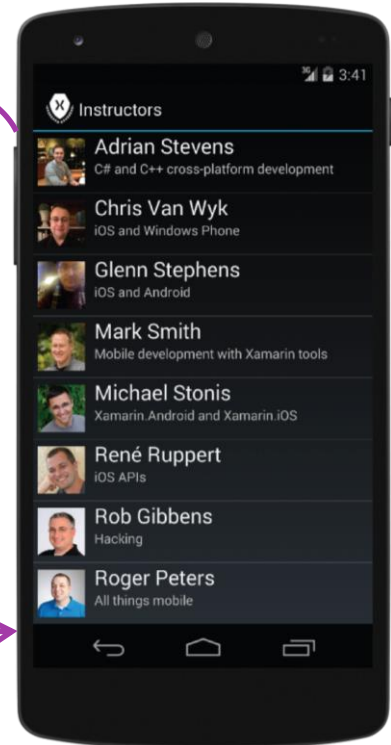
How to Reuse Inflated Layouts

❖ **GetView** will receive a layout in **convertView** to reuse if one is available

```csharp
public override View GetView(int position, View convertView, ViewGroup parent)
{
  var view = convertView;

  if (view == null)
  {
    view = context.LayoutInflater.Inflate(... );
  }
  ...
}
```

Only inflate a new layout if **ConvertView** is **null**

What is View.Tag?

❖ `View` has a **`Tag`** property you can use to store any extra info you need

```
public class View : ...
{
  public virtual Java.Lang.Object Tag { get; set; }
  ...
}
```

Your data must inherit from Java's **Object** base class

What is a View Holder?

❖ **ViewHolder** is the traditional name for a class that contains cached view references

```
public class ViewHolder : Java.Lang.Object
{
    public ImageView Photo      { get; set; }
    public TextView  Name       { get; set; }
    public TextView  Specialty { get; set; }
}
```

Inherits from Java's object so it can be stored in **View.Tag**

One property per view

How to Cache View References

❖ Cache view references in the layout's **Tag** so you only find references once when the layout is inflated, not each time the layout is reused

```csharp
public override View GetView(int position, View convertView, ViewGroup parent)
{ ...
  view = context.LayoutInflater.Inflate(...);

  var p = view.FindViewById<ImageView>(Resource.Id.photoImageView);
  var n = view.FindViewById<TextView >(Resource.Id.nameTextView);
  var s = view.FindViewById<TextView >(Resource.Id.specialtyTextView);

  view.Tag = new ViewHolder() { Photo = p, Name = n, Specialty = s };
  ...
}
```

Cache the reference during creation of the layout

Individual Exercise

Use layout recycling and the view-holder pattern

Xamarin University

Summary

1. Reuse inflated layouts to reduce memory usage

2. Cache view references to increase performance
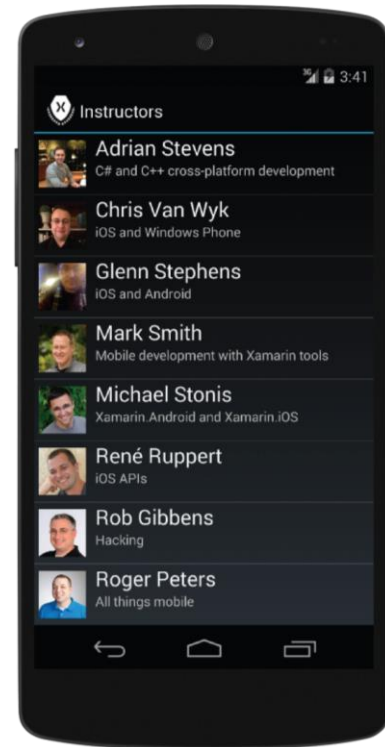
Enable fast scrolling and code a section indexer

Tasks

1. Enable `ListView` fast scrolling

2. Implement `ISectionIndexer` on a custom Adapter
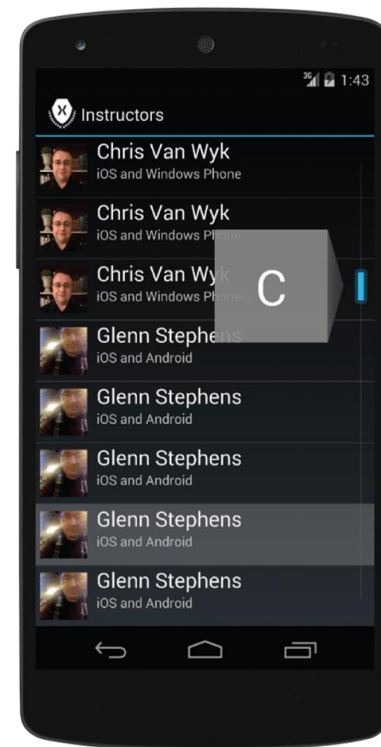
How to Enable Fast Scrolling

❖ Set the `ListView`'s `FastScrollEnabled` property to `true` to turn on fast scrolling
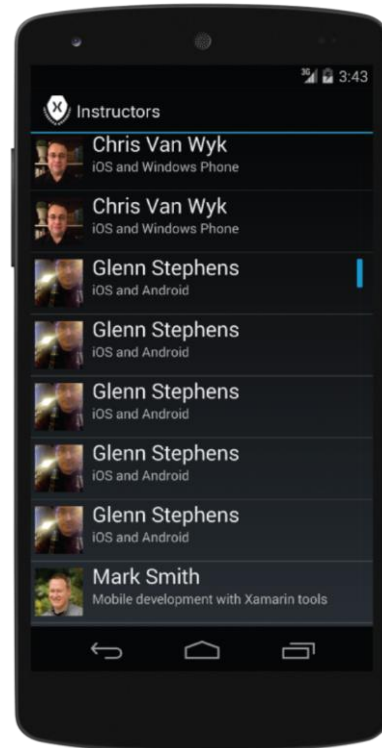
User can drag the *thumb* to scroll quickly (thumb only appears when the list contains multiple screens of data)

What is a Section?

❖ A *section* is a logical group in a list of data, you decide what the sections should be in your data

Data
Alex
Allen	"A" section
Ann
Carl
Carol	"C" section
Chris
Daisy	"D" section
Dave
Earl
Ed
Emily	"E" section
Erin
Evan
Frank	"F" section
Fred

What is a Section Indexer?

❖ A *Section Indexer* reports section labels and indices to a `ListView` to help the user navigate
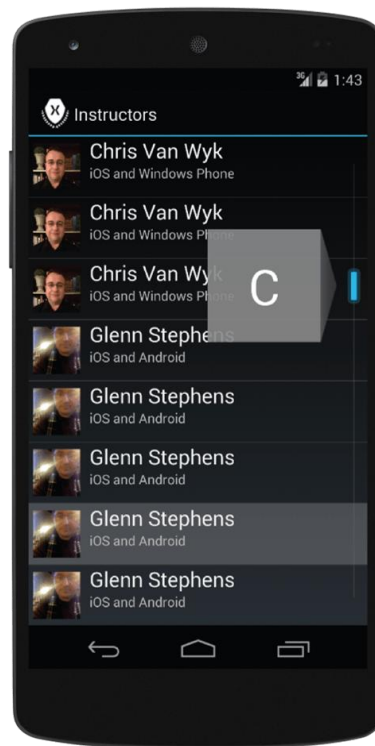
Section Indexer tells the `ListView` where the sections are and the label to display

How to Code a Section Indexer

❖ Implement **ISectionIndexer** on your Adapter, **ListView** checks for this interface and uses it if available

```csharp
public interface ISectionIndexer
{
    Java.Lang.Object[] GetSections();

    int GetPositionForSection(int section);
    int GetSectionForPosition(int position);
}
```

How to Code GetSections

❖ **`GetSections`** returns the section labels as an array of Java objects

Data	List position	Section index	Section label
Alex	0	0	A
Allen	1	0	A
Ann	2	0	A
Carl	3	1	C
Carol	4	1	C
Chris	5	1	C
Daisy	6	2	D
Dave	7	2	D
Earl	8	3	E
Ed	9	3	E
Emily	10	3	E
Erin	11	3	E
Evan	12	3	E
Frank	13	4	F

A	C	D	E	F

`GetSections` should return this array

How to Code GetPositionForSection

❖ Return the index of the first list position for the given section

Data	List position	Section index	Section label
Alex	0	0	A
Allen	1	0	A
Ann	2	0	A
Carl	3	1	C
Carol	4	1	C
Chris	5	1	C
Daisy	6	2	D
Dave	7	2	D
Earl	8	3	E
Ed	9	3	E
Emily	10	3	E
Erin	11	3	E
Evan	12	3	E
Frank	13	4	F

```
int GetPositionForSection(
      int section);
```

How to Code GetSectionForPosition

❖ Return the index of the section containing the given list position

Data	List position	Section index	Section label
Alex	0	0	A
Allen	1	0	A
Ann	2	0	A
Carl	3	1	C
Carol	4	1	C
Chris	5	1	C
Daisy	6	2	D
Dave	7	2	D
Earl	8	3	E
Ed	9	3	E
Emily	10	3	E
Erin	11	3	E
Evan	12	3	E
Frank	13	4	F

```
int GetSectionForPosition(
        int position);
```

Group Exercise

Enable fast scrolling and code a section indexer

Flash Quiz

Flash Quiz

① Which **ListView** event is raised when the user clicks on a row?

    a) `Click`

    b) `ItemClick`

    c) `ItemSelected`

Flash Quiz

① Which **ListView** event is raised when the user clicks on a row?

   a) `Click`

   b) <u>ItemClick</u>

   c) `ItemSelected`

Flash Quiz

② What is *inflation*?

    a) Populating a list with rows

    b) Creating a `Drawable` from an Asset file

    c) Loading code-behind data into the views of a row

    d) Creating a view hierarchy from a layout file

Flash Quiz

② What is *inflation*?

    a) Populating a list with rows

    b) Creating a `Drawable` from an Asset file

    c) Loading code-behind data into the views of a row

    d) <u>Creating a view hierarchy from a layout file</u>

Flash Quiz

③ If you implement the *view-holder pattern* correctly, how many times will you use `FindViewById` to locate each view in a row's view hierarchy?

   a) 0

   b) 1

   c) 2

Flash Quiz

③ If you implement the *view-holder pattern* correctly, how many times will you use `FindViewById` to locate each view in a row's view hierarchy?

a) 0

b) 1

c) 2

Flash Quiz

④ To provide indexing, you implement **ISectionIndexer** on which class?

    a) The **ListView** itself

    b) Your custom Adapter

    c) Your Main Activity

Flash Quiz

④ To provide indexing, you implement **ISectionIndexer** on which class?

    a) The **ListView** itself

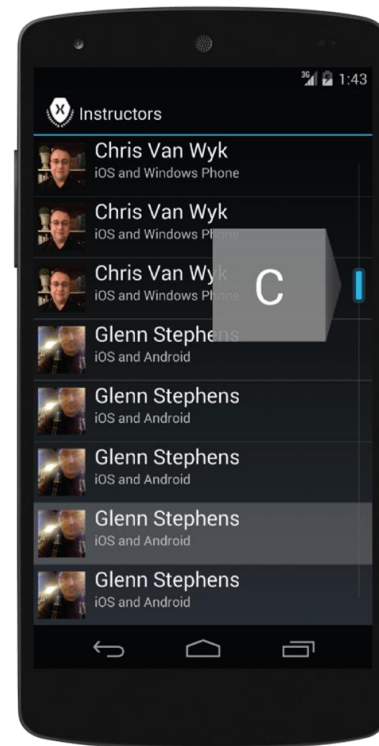   b) <u>Your custom Adapter</u>

    c) Your Main Activity

Flash Quiz

⑤ **`GetSectionForPosition`** maps indices from _____

    a) list position to section index

    b) section index to list position

    c) view position to list position

Flash Quiz

⑤ `GetSectionForPosition` maps indices from _____
    a) <u>list position to section index</u>
    b) section index to list position
    c) view position to list position

Summary

1. Enable `ListView` fast scrolling
2. Implement `ISectionIndexer` on a custom Adapter