AND115

RecyclerView and CardView in Android

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, Xamarin Studio, and Visual Studio are either registered trademarks or trademarks of Microsoft in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

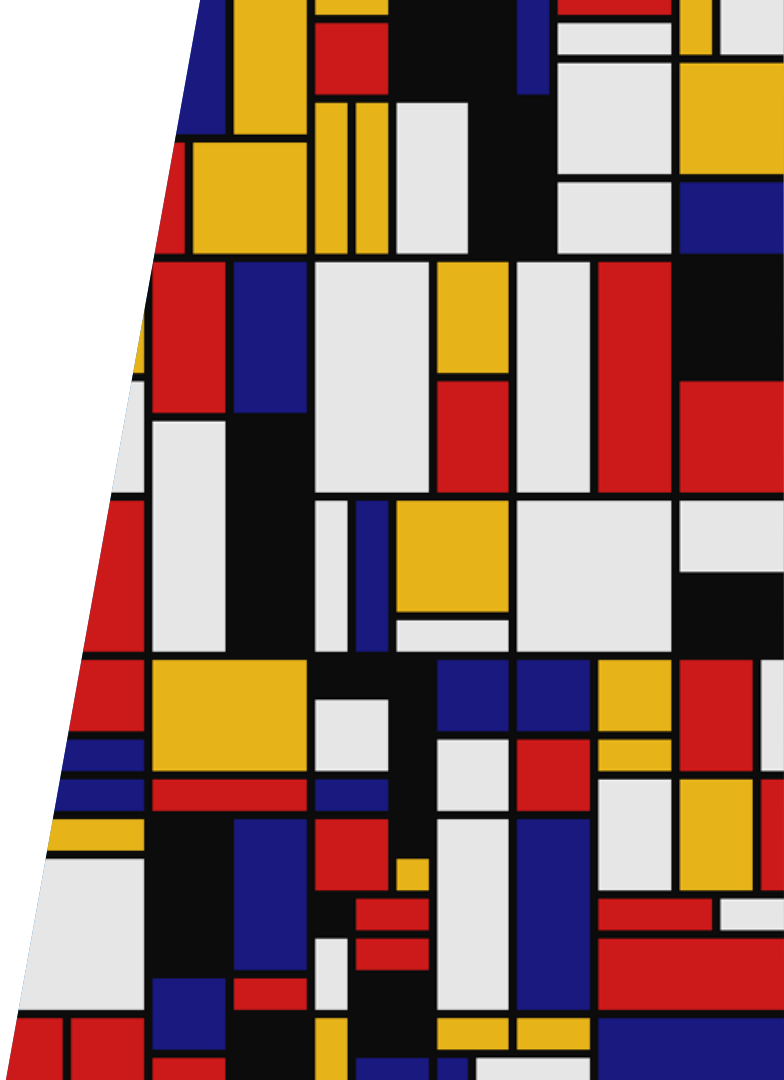
Objectives

1. Display a collection using `RecyclerView`
2. Update the UI after a data change
3. Respond to user actions
4. Show data in a `CardView`

Display a collection using RecyclerView

Tasks

1. Create a `RecyclerView`
2. Select a layout manager
3. Code an item-layout file
4. Code a view holder
5. Code an adapter



Motivation [collections]

- Apps often need to display collections of data

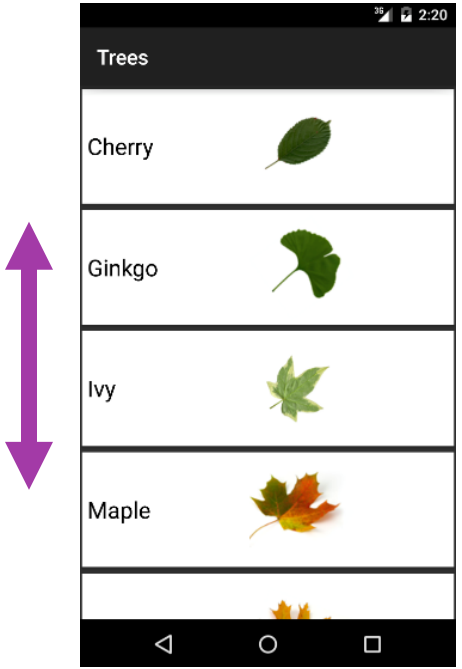
Typical variations:

- vertical or horizontal
- list or grid

Motivation [efficiency]

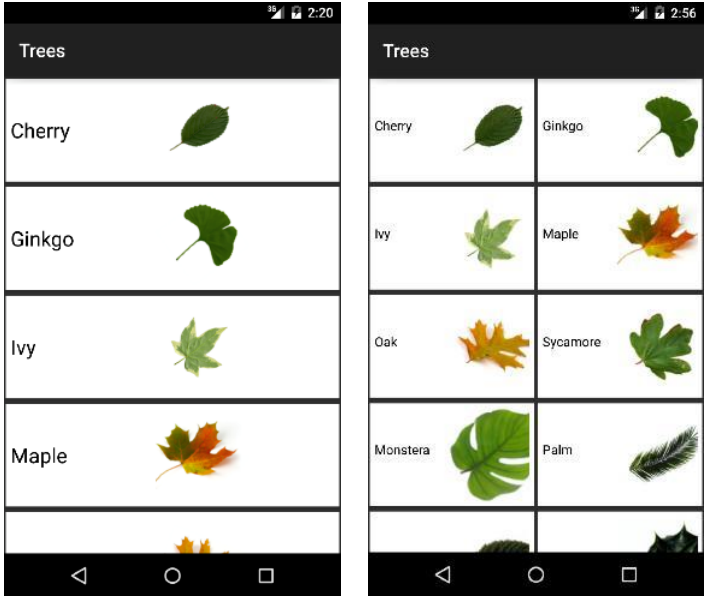
- Collections are often too large to display the entire dataset at once, items must be scrolled into view

App must handle this efficiently to get smooth scrolling and minimize pressure on the garbage collector



What is RecyclerView?

- **`RecyclerView`** displays a collection, it is optimized to handle large datasets efficiently by reusing views and requiring view-holders
- Handles scrolling and view recycling
- Pluggable layout policy to support various layouts



RecyclerView vs. legacy views

- **`RecyclerView`** can generally replace both **`ListView`** and **`GridView`**, but does not offer exactly the same features

	`RecyclerView`	`ListView/GridView`
List or grid layout	yes	yes
View recycling	yes	yes
Add/remove animations	yes	no
View-holder pattern	required	optional
Item click event	no	yes
Predefined adapters	no	yes
Fast scroll/indexer	no	yes

RecyclerView packaging

- **`RecyclerView`** is in a *support library* which must be added using the Xamarin Component Store, or Nuget (preferred)

1. Add the Xamarin Component or the NuGet package

```
<android.support.v7.widget.RecyclerView>
  ...
</android.support.v7.widget.RecyclerView>
```

2. Qualify the name

Group Exercise

Create a RecyclerView

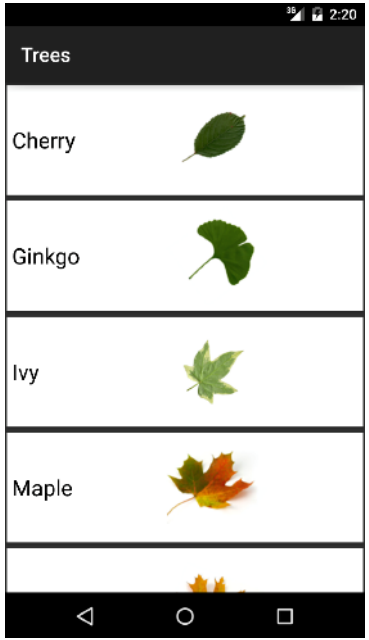
Xamarin University

Architecture

- Displaying a collection is a collaboration among several classes

`RecyclerView`
handles scrolling and manages a pool of views

`LayoutManager`
positions items



`MyLayout.axml`
defines item layout

`ViewHolder`
stores view references, detects item-click

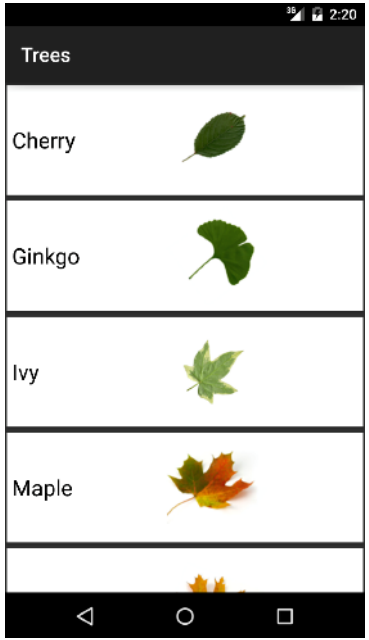
`Adapter`
inflates layout, binds data to views, reports item-click

Architecture

- Displaying a collection is a collaboration among several classes

`RecyclerView`
handles scrolling and manages a pool of views

`LayoutManager`
positions items



`MyLayout.axml`
defines item layout

`ViewHolder`
stores view references, detects item-click

`Adapter`
inflates layout, binds data to views, reports item-click

Your code

What is a layout manager?

- A *layout manager* arranges your items in the **`RecyclerView`**

Layout manager calculates the size and position of each item

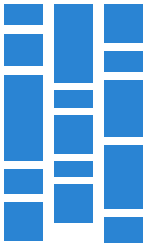
Predefined layout managers

- Android supplies layout managers for a few common **layout styles**

`LinearLayoutManager`

`GridLayoutManager`

`StaggeredGridLayoutManager`



LinearLayoutManager [overview]

- **`LinearLayoutManager`** arranges your items in a single column or row

```
var lm = new LinearLayoutManager(this, LinearLayoutManager.Vertical, false);
```

Context

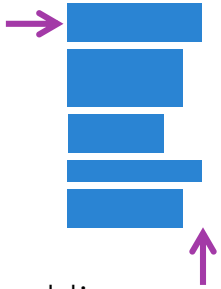
Orientation

Reverse (e.g. in a vertical list, it would layout items from bottom to top)

LinearLayoutManager [item size]

- **`LinearLayoutManager`** lets you keep items uniform or have them vary

In a vertical list, row height is based on item height and can vary



In a vertical list, row width is taken from the width of the containing **`RecyclerView`**, your item does not need to occupy the entire row, but it will be forced to the row width if it is too large

GridLayoutManager [overview]

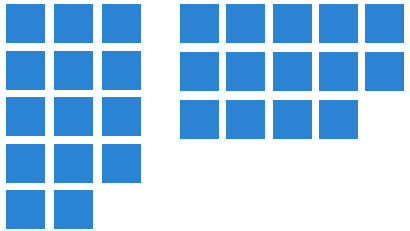
- **`GridLayoutManager`** arranges your items in a grid

Number of spans (e.g. number of columns for a vertical grid)

```
var lm = new GridLayoutManager(this, 3, GridLayoutManager.Vertical, false);
```

Context

Orientation



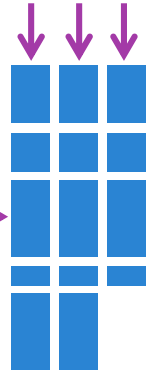
Reverse (e.g. in a vertical grid, it would layout items from bottom to top)

GridLayoutManager [default sizing]

- **`GridLayoutManager`** distributes the space uniformly along one axis while the other axis can vary based on the size of your items

In a vertical grid, items are forced to uniform width

In a vertical grid, row heights can vary, height is taken from the tallest item in each row



GridLayoutManager [spans]

- **`GridLayoutManager`** allows items to occupy multiple spans

You write this class and load an instance into a grid layout manager

```
public class MySpanLookup : GridLayoutManager.SpanSizeLookup
{
  public override int GetSpanSize(int position) { ... }
}
```

Your code decides the span count for each of your items

Span size is 2

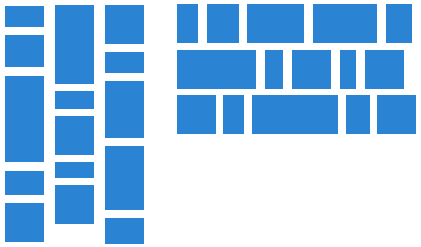
StaggeredGridLayoutManager [overview]

- **`StaggeredGridLayoutManager`** arranges your items in a compact grid

```
var sglm = new StaggeredGridLayoutManager(3, StaggeredGridLayoutManager.Vertical);
```

Number of spans (e.g. number of columns for a vertical grid)

Orientation

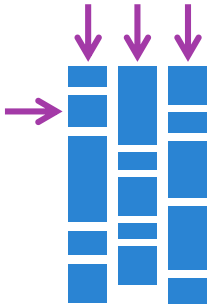


StaggeredGridLayoutManager [sizing]

- **`StaggeredGridLayoutManager`** only modifies your item size in one dimension

Items are forced to uniform width in a vertical layout

Items keep their natural height in a vertical layout, the rows are not clearly delineated since the items are not forced to be the same height



StaggeredGridLayoutManager [full span]

- **`StaggeredGridLayoutManager`** allows items to occupy an entire axis

```
public override void OnBindViewHolder(RecyclerView.ViewHolder holder, int position)
{ ...
  var lp = holder.ItemView.LayoutParameters.JavaCast<StaggeredGridLayoutManager.LayoutParams>();
  if (...)
    lp.FullSpan = true;
  else
    lp.FullSpan = false;
  ...
}
```

Set to full-span in your adapter using the layout parameters

How to set a layout manager

- You create a layout manager instance and set it in the **RecyclerView** (there is no default, you will get an exception if you do not load one)

```
var lm = new LinearLayoutManager(this, LinearLayoutManager.Vertical, false);
var rv = FindViewById<RecyclerView>(Resource.Id.recyclerView);

rv.SetLayoutManager(lm);
```

Load your selected layout type

Group Exercise

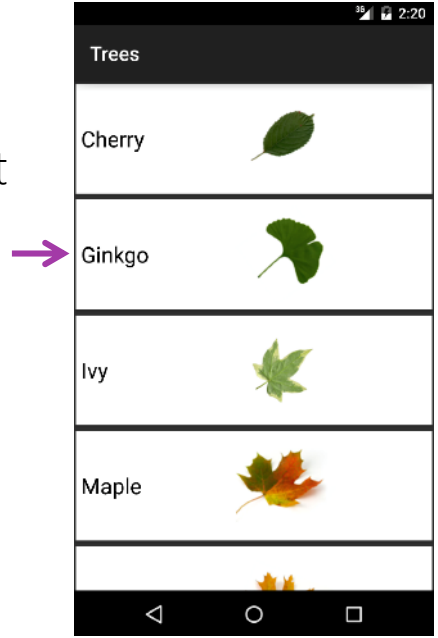
Set a layout manager

Xamarin University

Motivation

- You get to decide how to display your data items in the UI

E.g. you might show one piece of text and an image

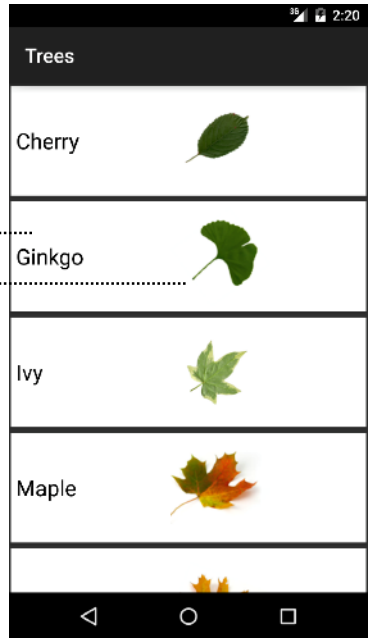


What is an item-layout file?

- An *item-layout file* defines the view hierarchy that will display one of your data items

```
<LinearLayout ... >
  <TextView   ... />
  <ImageView  ... />
</LinearLayout>
```

You write this XML



How to code an item-layout file?

- You write an XML layout file and include it as layout Resource in your project

MyLayout.axml

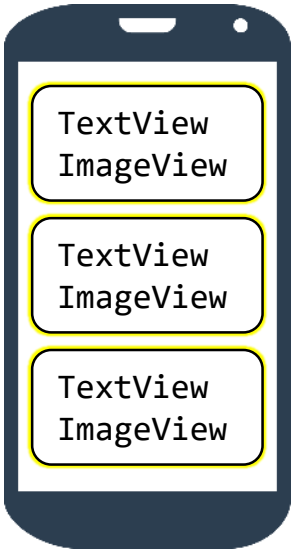
```
<LinearLayout ... >
  <TextView  android:id="@+id/textView"  ... />
  <ImageView android:id="@+id/imageView" ... />
</LinearLayout>
```

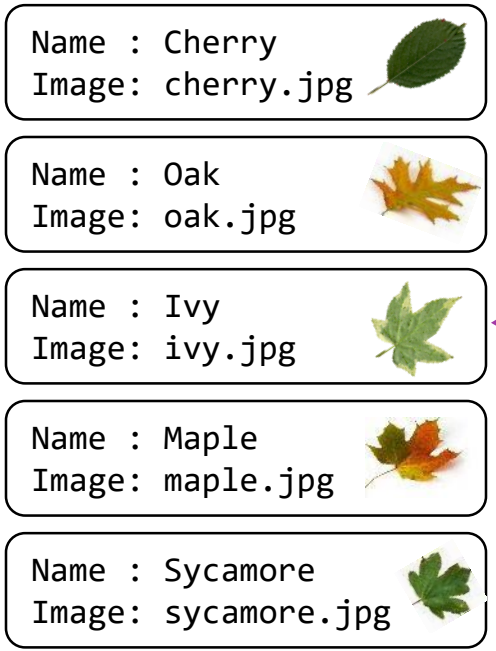
Add an Id to the views you will need to access from code

Memory efficiency

- For efficiency, **`RecyclerView`** only instantiates the item-layout file for visible items

Only 3 items fit on screen so only 3 copies are needed

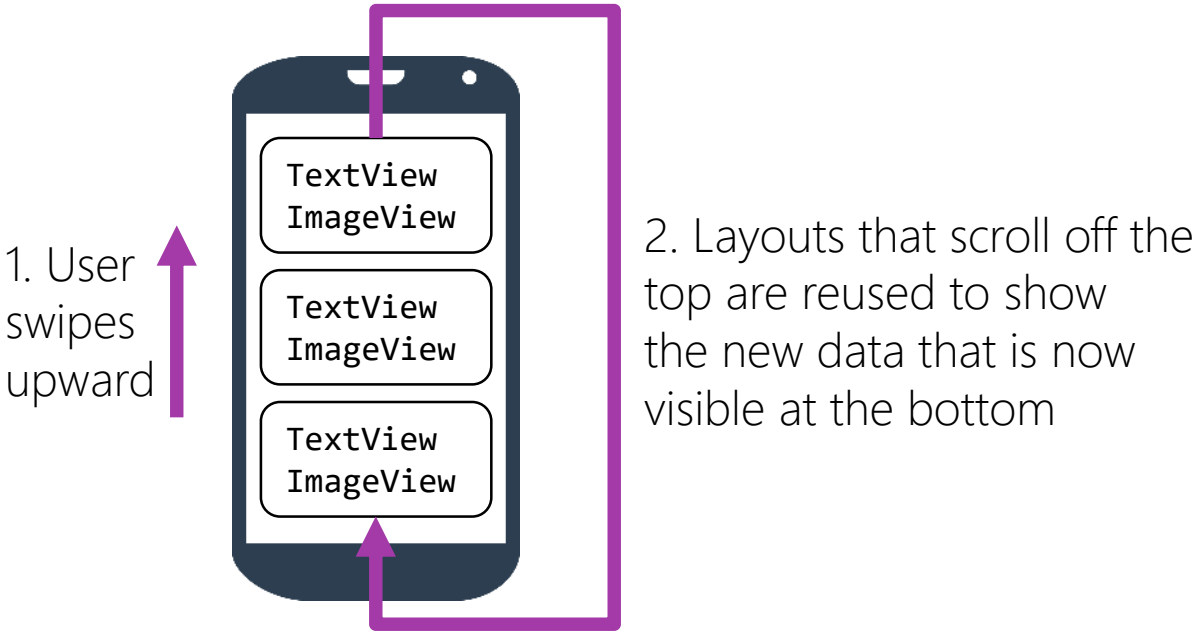




Data set is larger than fits on screen, no need to allocate one layout for each

Layout recycling

- For efficiency, **`RecyclerView`** reuses instantiated item-layout files as the user scrolls



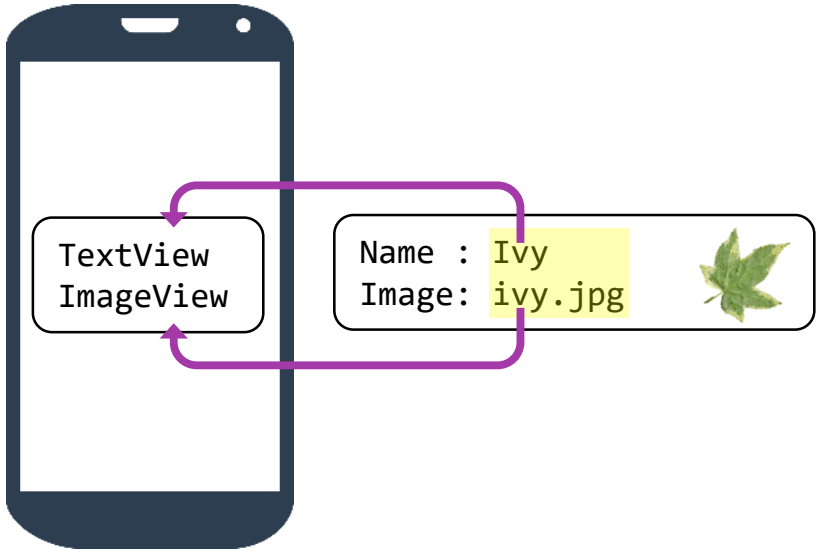
Group Exercise

Code an item-layout file

Motivation [binding]

- You need to load your data into your UI views

E.g. You must load **`Name`** into the **`TextView`**'s **`Text`** property



Motivation [view lookup]

- Must get references to the views in your layout file to load data

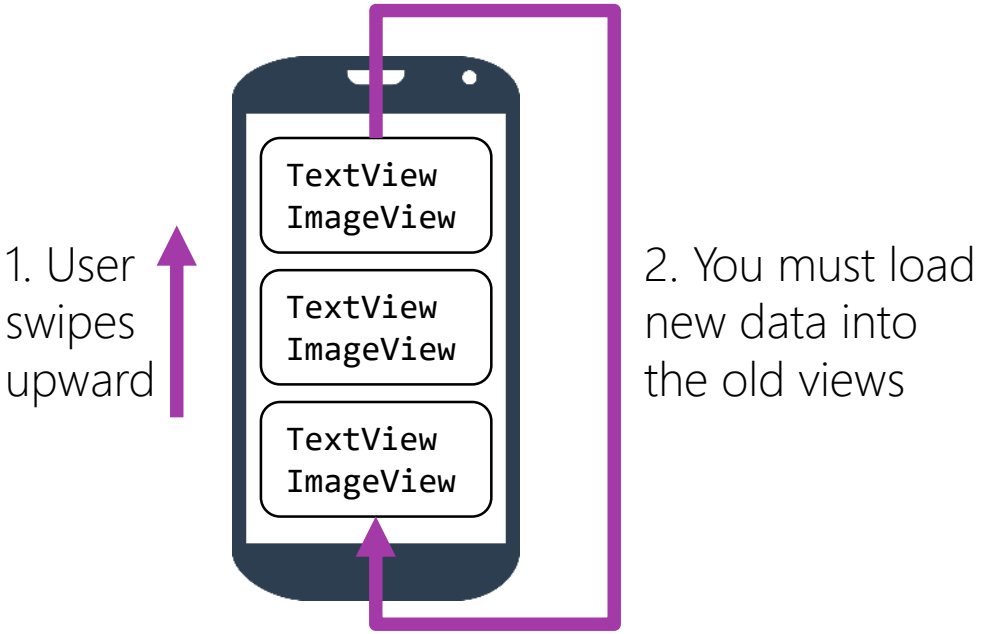
```
<LinearLayout ... >
  <TextView  android:id="@+id/textView"  ... />
  <ImageView android:id="@+id/imageView" ... />
</LinearLayout>
```

```
var tv = layout.FindViewById<TextView >(Resource.Id.textView );
var iv = layout.FindViewById<ImageView>(Resource.Id.imageView);
```

Lookup the individual views from your layout file by Id

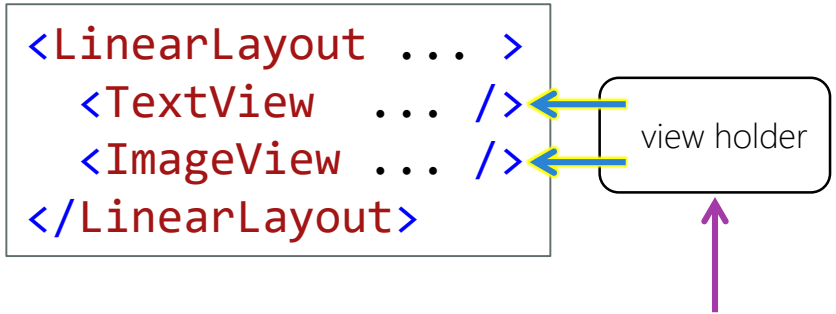
Motivation [efficiency]

- Should avoid calling **`FindViewById`** every time **`RecyclerView`** reuses a layout



What is a view holder?

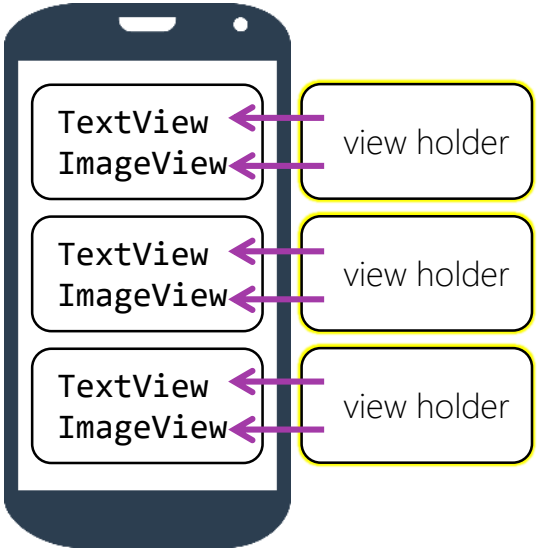
- A *view holder* is an object that stores references to the views in your item-layout file



The view-holder is given an inflated layout file and uses **`FindViewById`** to get references to the views inside

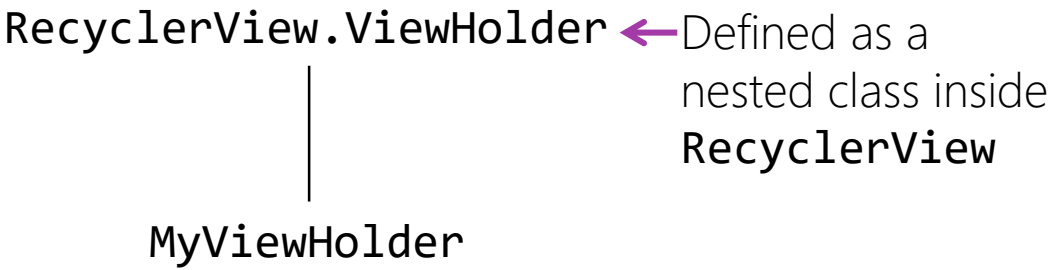
View holder instances

- You will have one view-holder for each instantiated layout file



View holder base class

- Your view holder must derive from `RecyclerView.ViewHolder`



View holder base class services

- **`RecyclerView.ViewHolder`** stores a reference to the UI for the item, you set this property using the constructor

Reference to the layout-file instance

```
public abstract class ViewHolder : Java.Lang.Object
{ ...
  public ViewHolder(View itemView) { ... }

  public View ItemView { get; set; }
}
```

How to code a view holder

- Your view holder should provide references to each internal view, these are typically set in its constructor

Reference to the inflated layout file, stored in base class

itemView is the inflated layout file, use **FindViewById** to locate views inside it

```
public class MyViewHolder : RecyclerView.ViewHolder
{
  public MyViewHolder(View itemView)
  : base(itemView)
  {
    Name  = itemView.FindViewById<TextView >(Resource.Id.textView );
    Image = itemView.FindViewById<ImageView>(Resource.Id.imageView);
  }

  public TextView  Name  { get; private set; }
  public ImageView Image { get; private set; }
}
```

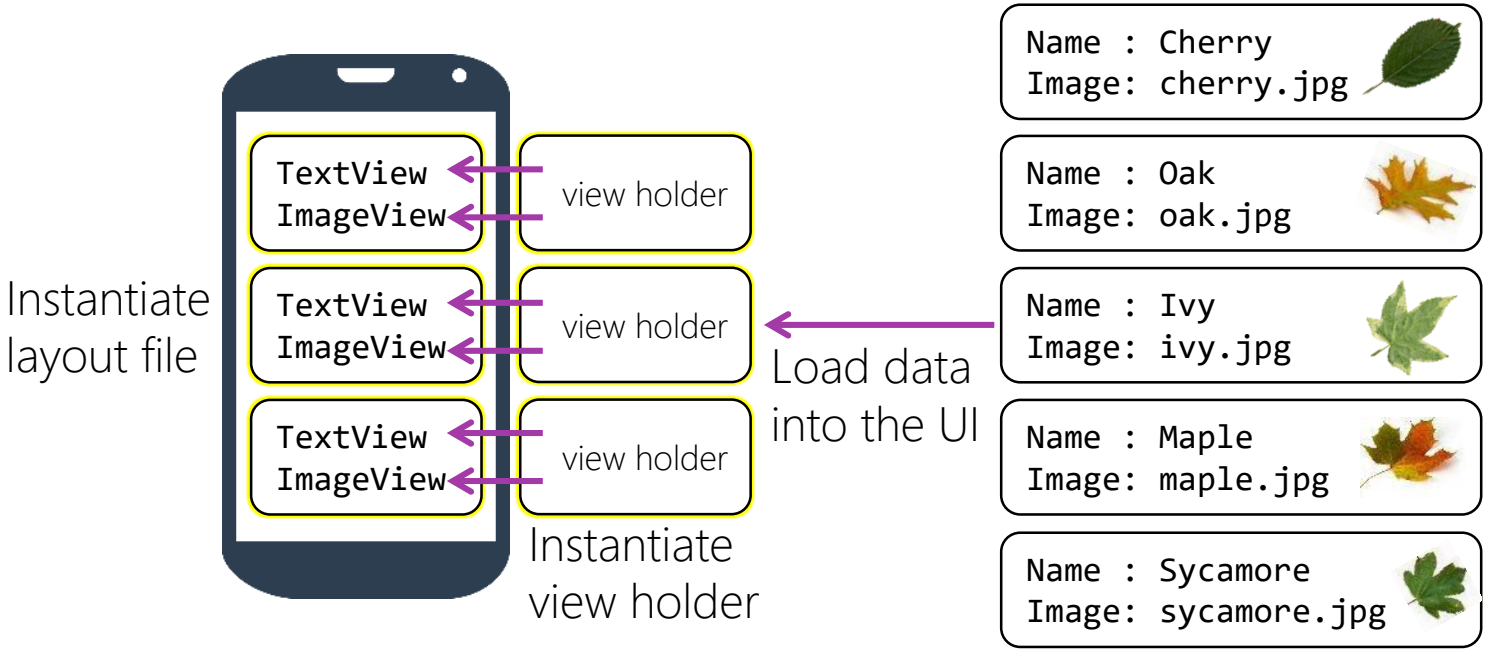
Group Exercise

Code a view holder

Xamarin University

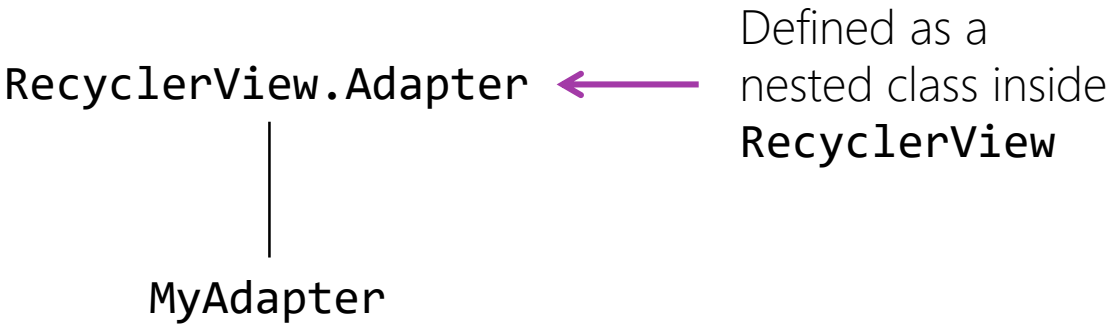
What is an adapter?

- An *adapter* is responsible for creating and populating the UI



Adapter base class

- Your adapter must derive from **`RecyclerView.Adapter`**



Adapter base class services

- **`RecyclerView.Adapter`** declares the abstract members you must code in your adapter

Number of data items

Instantiate the layout file and view holder

```
public abstract class Adapter : Java.Lang.Object
{ ...
  public abstract int ItemCount { get; }

  public abstract RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup parent, int viewType);

  public abstract void OnBindViewHolder(RecyclerView.ViewHolder holder, int position);
}
```

Load the data at the given position into the UI stored in the given view holder

Data access

- Your adapter needs access to your data set so it can populate the UI and provide the item count

Your data →

```
public class Tree
{ ...
  public string   Name  { get; set; }
  public Drawable Image { get; set; }
}
```

Typical to pass to the constructor →

```
public class MyAdapter : RecyclerView.Adapter
{ ...
  List<Tree> myData;

  public MyAdapter(List<Tree> data)
  {
    this.myData = data;
  }
}
```

How to code ItemCount

- Your adapter must report the item count, used by the **RecyclerView**

Abstract member from base class, must override

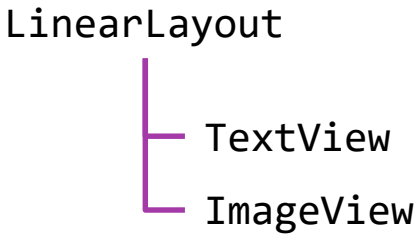
```
public class MyAdapter : RecyclerView.Adapter
{
  List<Tree> myData;

  public override int ItemCount
  {
    get { return myData.Count; }
  }
  ...
}
```

What is Inflation?

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

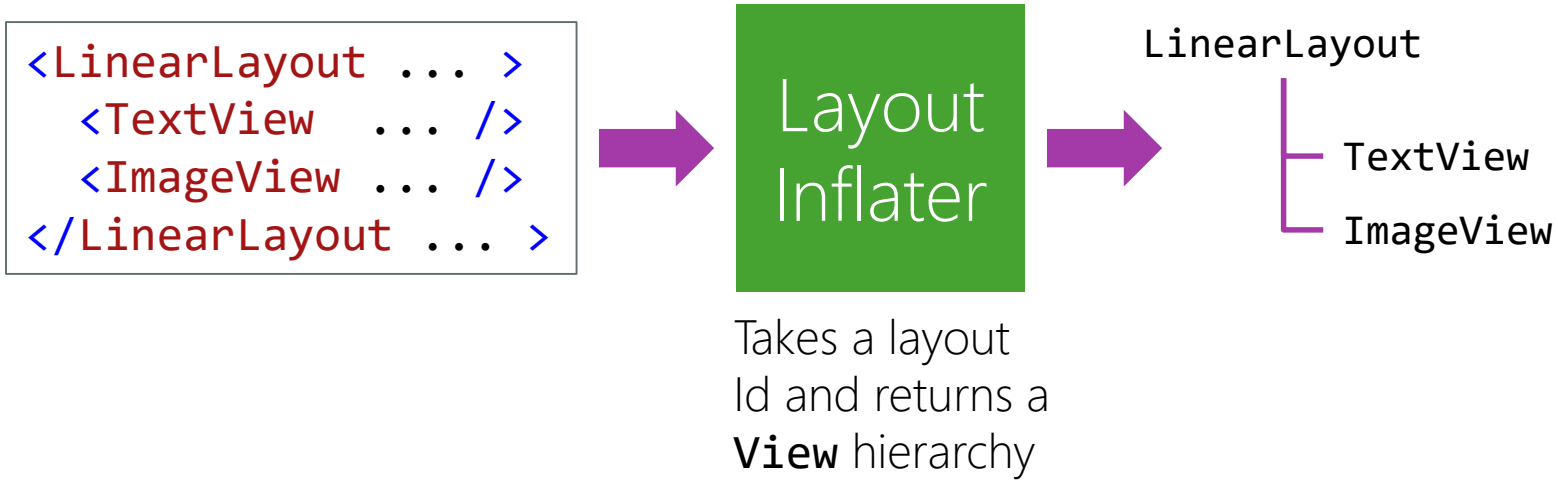
```
<LinearLayout ... >
  <TextView  ... />
  <ImageView ... />
</LinearLayout ... >
```

Inflation creates a view hierarchy from a layout file



What is a LayoutInflater?

- Library class **`LayoutInflater`** performs inflation



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 use the parent view passed to **`OnCreateViewHolder`** to get one

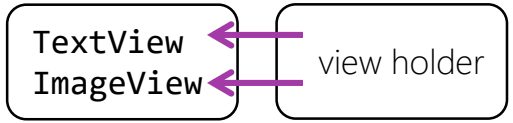
The **`ViewGroup`** that will contain your inflated layout

```
public class MyAdapter : RecyclerView.Adapter
{
  public override RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup parent, int viewType)
  {
    var inflater = LayoutInflater.From(parent.Context);
    ...
  }
  ...
}
```

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

How to code OnCreateViewHolder

- **`OnCreateViewHolder`** inflates a layout and creates a view holder



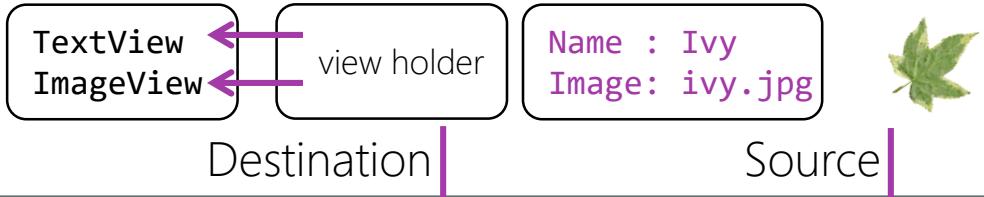

```
public class MyAdapter : RecyclerView.Adapter
{ ...
  List<Tree> myData;

  public override RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup parent, int viewType)
  {
    var inflater = LayoutInflater.From(parent.Context);
    var view = inflater.Inflate(Resource.Layout.MyLayout, parent, false);
    return new MyViewHolder(view);
  }
}
```

2. Create a view holder

1. Inflate the item-layout file

How to code OnBindViewHolder

- **`OnBindViewHolder`** copies the data into the UI



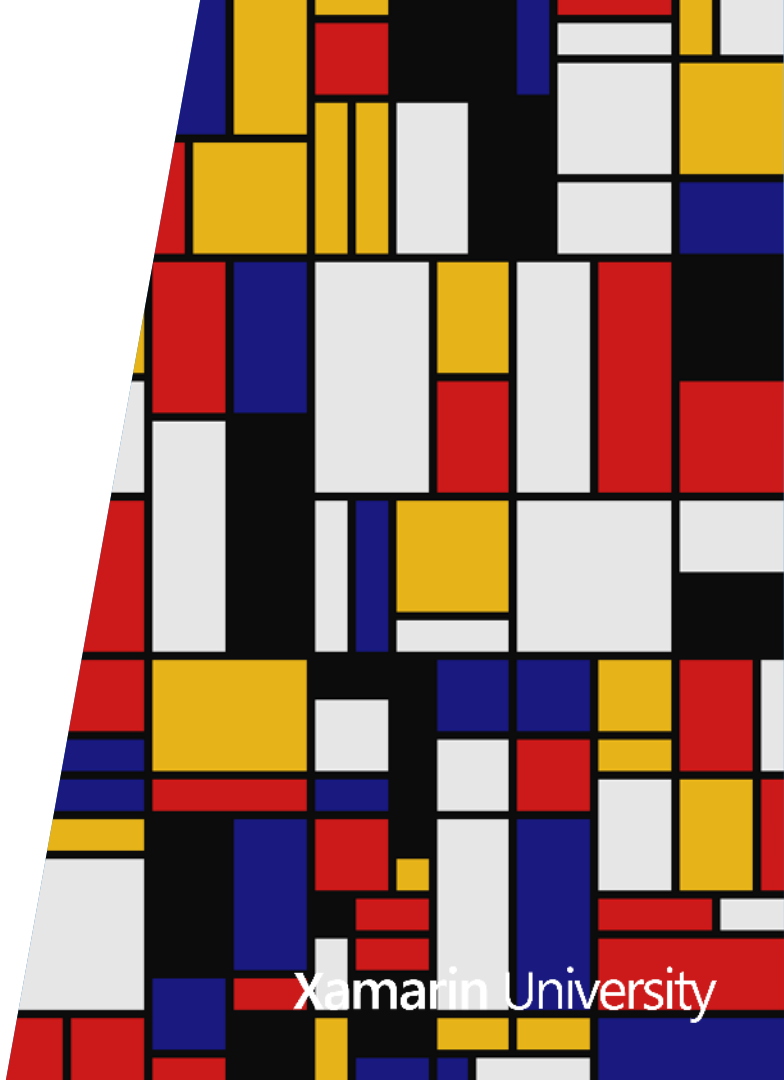

```
public class MyAdapter : RecyclerView.Adapter
{ ...
  List<Tree> myData;

  public override void OnBindViewHolder(RecyclerView.ViewHolder holder, int position)
  {
    var vh = (MyViewHolder)holder;

    vh.Name.Text = myData[position].Name;
    vh.Image.SetImageDrawable(myData[position].Image);
  }
}
```

Group Exercise

Code an adapter

Summary

1. Create a `RecyclerView`
2. Select a layout manager
3. Code an item-layout file
4. Code a view holder
5. Code an adapter



Update the UI after a data change

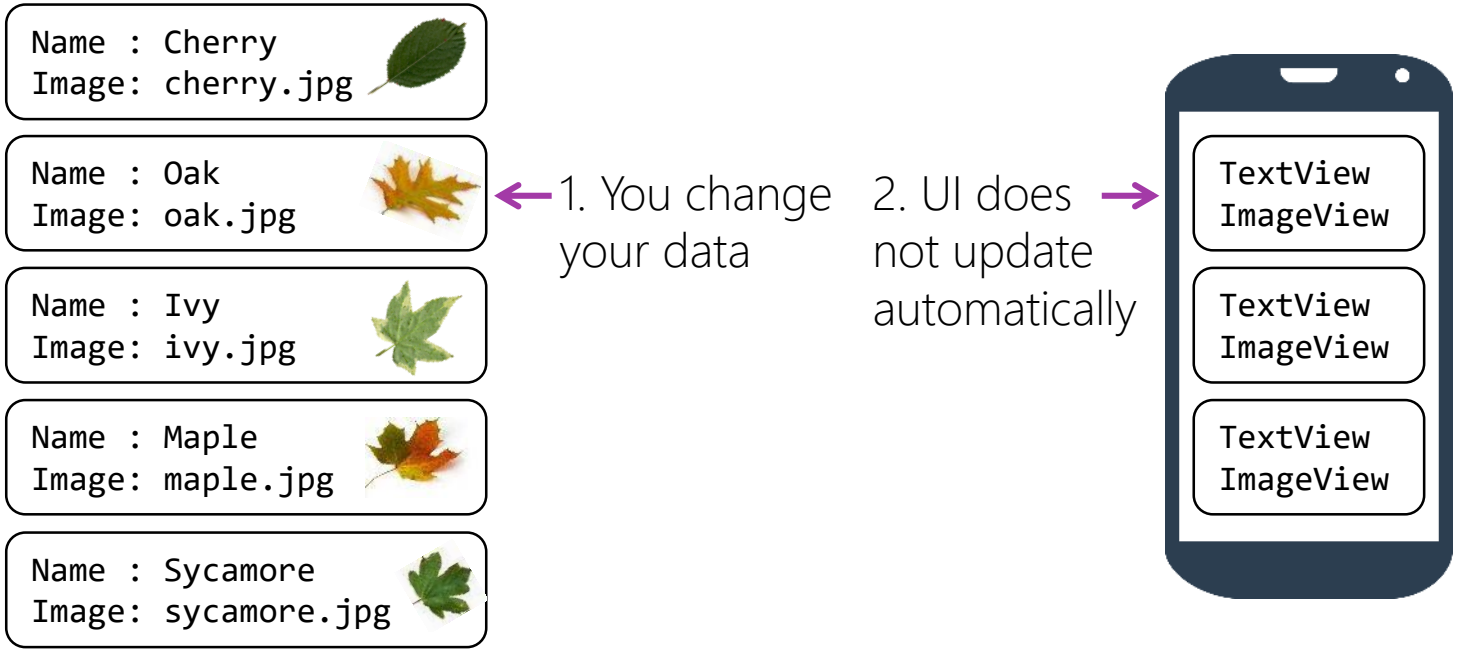
Xamarin University

Tasks

1. Notify `RecyclerView` when your data changes

Motivation

- **`RecyclerView`** does not know when your data changes

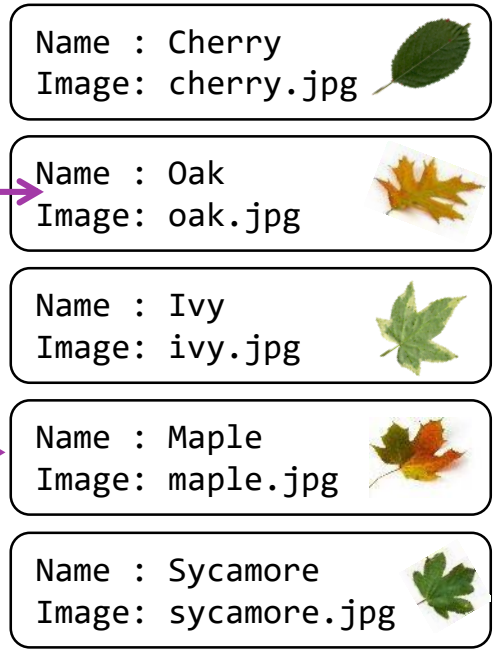


Types of changes

- There are two **types of changes** that might happen to your data

Values within an item are modified (called an *item change*)

Items are added, deleted, or moved (called a *structural change*)



Adapter methods

- You use methods from your adapter's base class to notify **`RecyclerView`** that your data has changed

Item changes

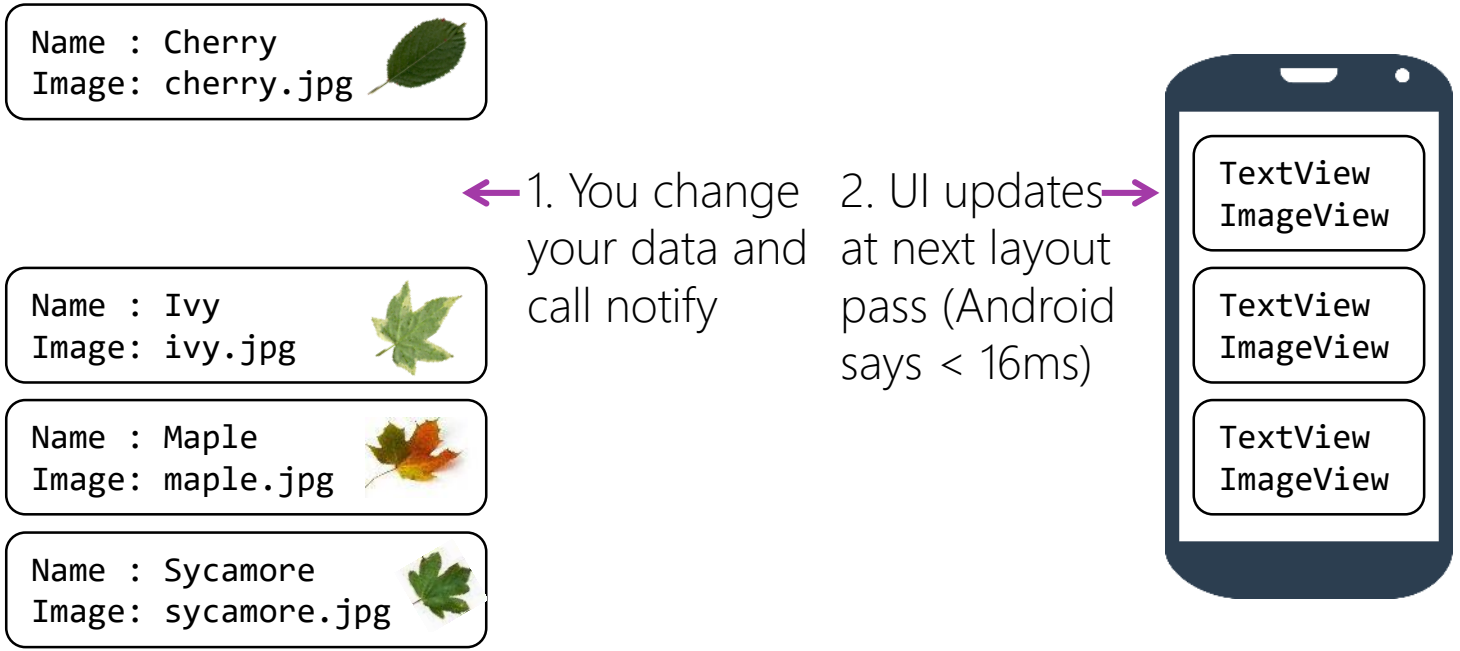
Structural changes

Force a full update

```
public abstract class Adapter : Java.Lang.Object
{ ...
  public void NotifyItemChanged      (int position);
  public void NotifyItemRangeChanged (int positionStart, int itemCount);

  public void NotifyItemInserted     (int position);
  public void NotifyItemRangeInserted(int positionStart, int itemCount);
  public void NotifyItemRemoved      (int position);
  public void NotifyItemRangeRemoved (int positionStart, int itemCount);
  public void NotifyItemMoved        (int fromPosition,  int toPosition);

  public void NotifyDataSetChanged();
}
```

Asynchronous update

- The UI updates asynchronously after you call one of the notify methods



Flash Quiz

Xamarin University

Flash Quiz

① Why are there so many notification methods for data changes?

a) Efficiency

b) To ensure the correct UI elements are updated

c) To encourage you to make all your data read-only

Flash Quiz

① Why are there so many notification methods for data changes?

a) Efficiency

b) To ensure the correct UI elements are updated

c) To encourage you to make all your data read-only

Flash Quiz

② When should you use the `NotifyDataSetChanged()` method?

a) When you have both item and structural changes
b) When you have more than five changes
c) You probably should not use it

Flash Quiz

② When should you use the `NotifyDataSetChanged()` method?

a) When you have both item and structural changes

b) When you have more than five changes

c) You probably should not use it

Summary

1. Notify **`RecyclerView`** when your data changes

Respond to user actions

Xamarin University

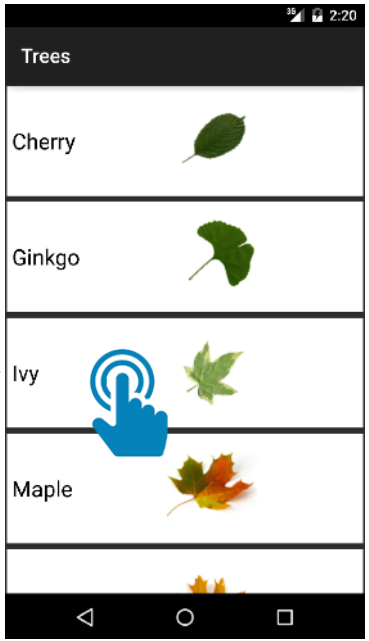
Tasks

1. Determine the position of the clicked item
2. Detect user actions
3. Report user actions via an event

Motivation

- Your app would like to be notified when the user touches an item, but **`RecyclerView`** does not offer an item-click event

You need to write code to detect a touch



Who implements item-click?

- You have to implement item-click manually; generally, in your View Holder and Adapter

`ViewHolder`

Detects item-click

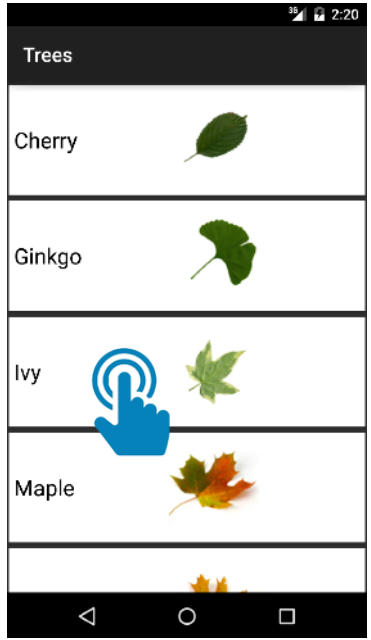
`Adapter`

Reports item-click

Event data

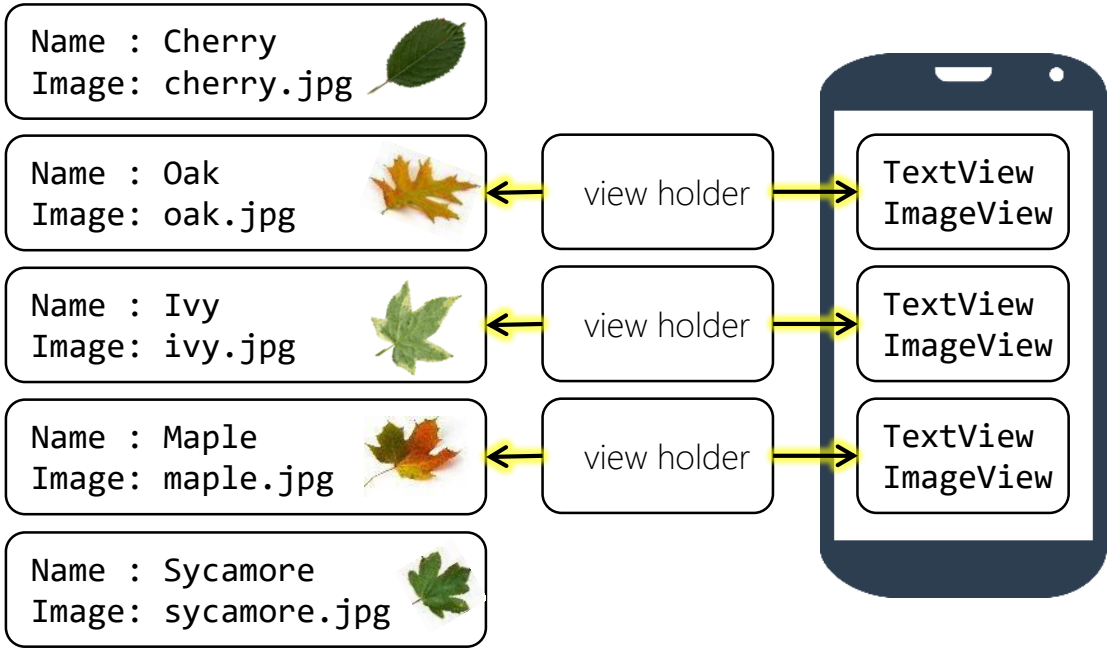
- Item-click events generally report the position of the clicked item

Your event handler needs to know which item was touched in order to process the event



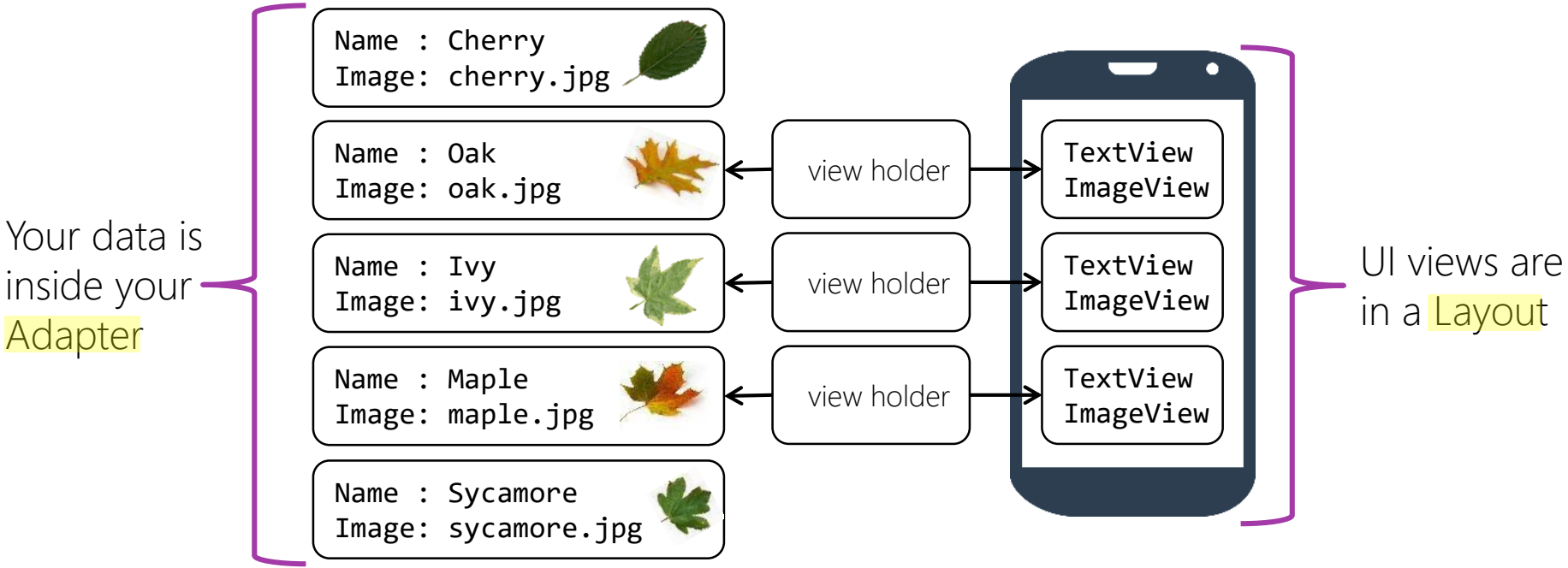
View Holder as connector

- A view holder instance is a **connector** between a data item and its UI



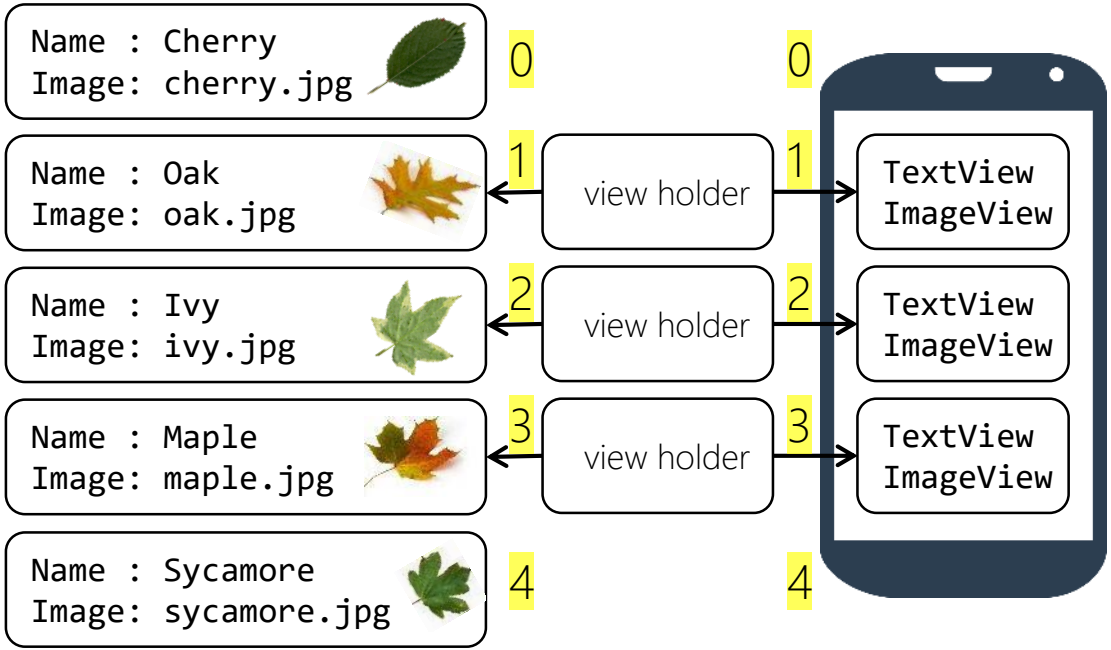
Adapter and Layout

- Your data and its UI are **hosted inside different things**



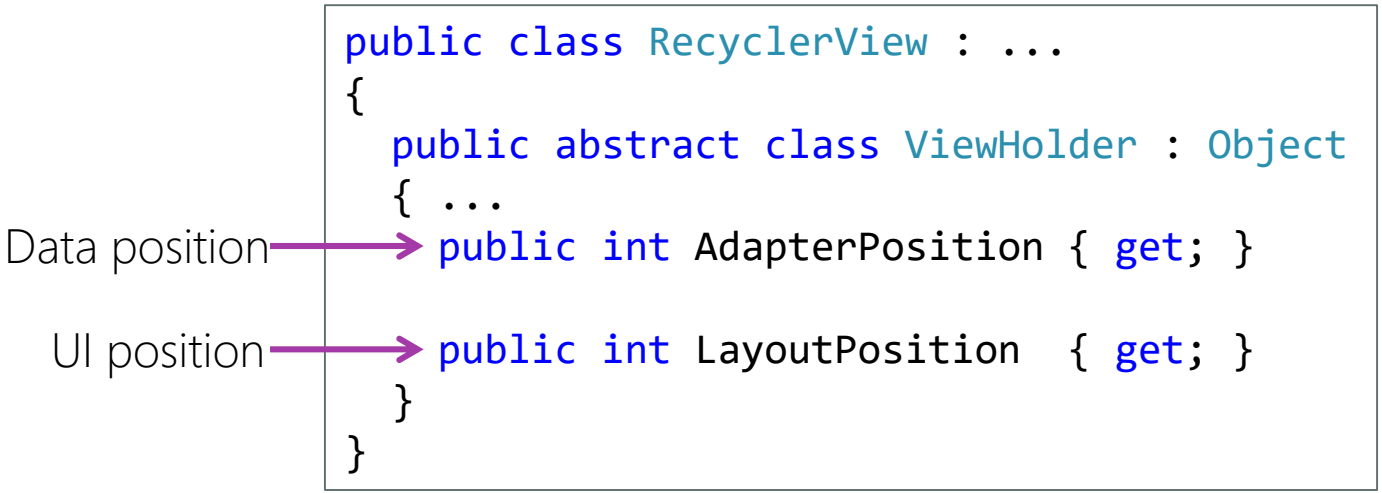
Two positions

- Each item has **two position values**, one for the data and one for the UI (the two values are the same most of the time)



View Holder position properties

- **`ViewHolder`** has properties for both position values




```
public class RecyclerView : ...
{
  public abstract class ViewHolder : Object
  { ...
    public int AdapterPosition { get; }

    public int LayoutPosition  { get; }
  }
}
```

Guidance

- Android provides general guidance on the role of the two position values

`AdapterPosition`

"...when writing an RecyclerView.Adapter, you probably want to use adapter positions..."

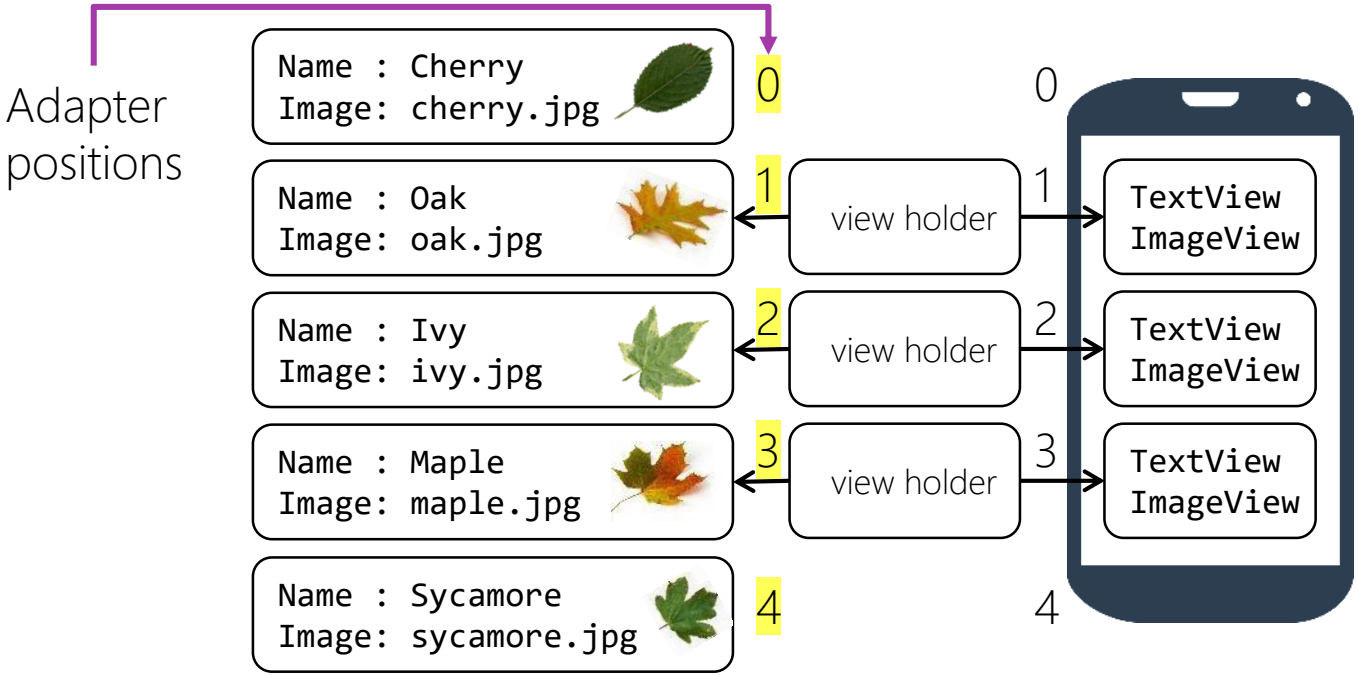
`LayoutPosition`

"...when writing a RecyclerView.LayoutManager you almost always want to use layout positions..."

We will use **`AdapterPosition`** - the details of **`LayoutPosition`** are not covered

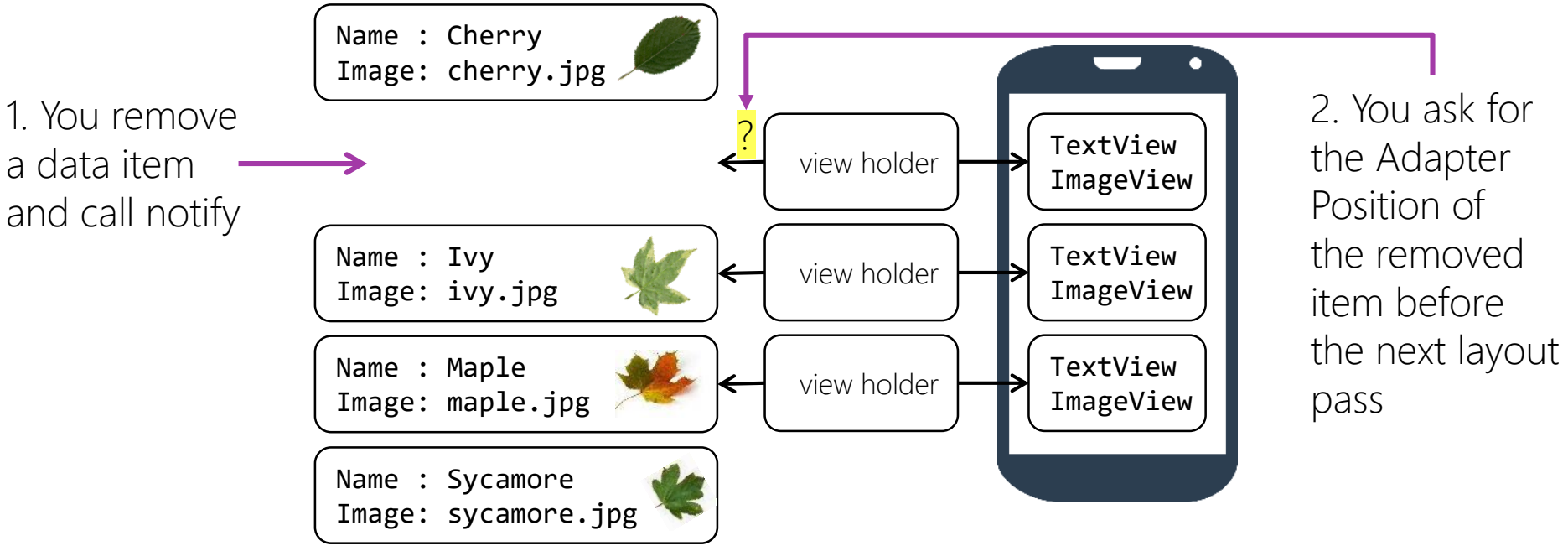
What is Adapter Position?

- *Adapter Position* is the position of an item in your data set



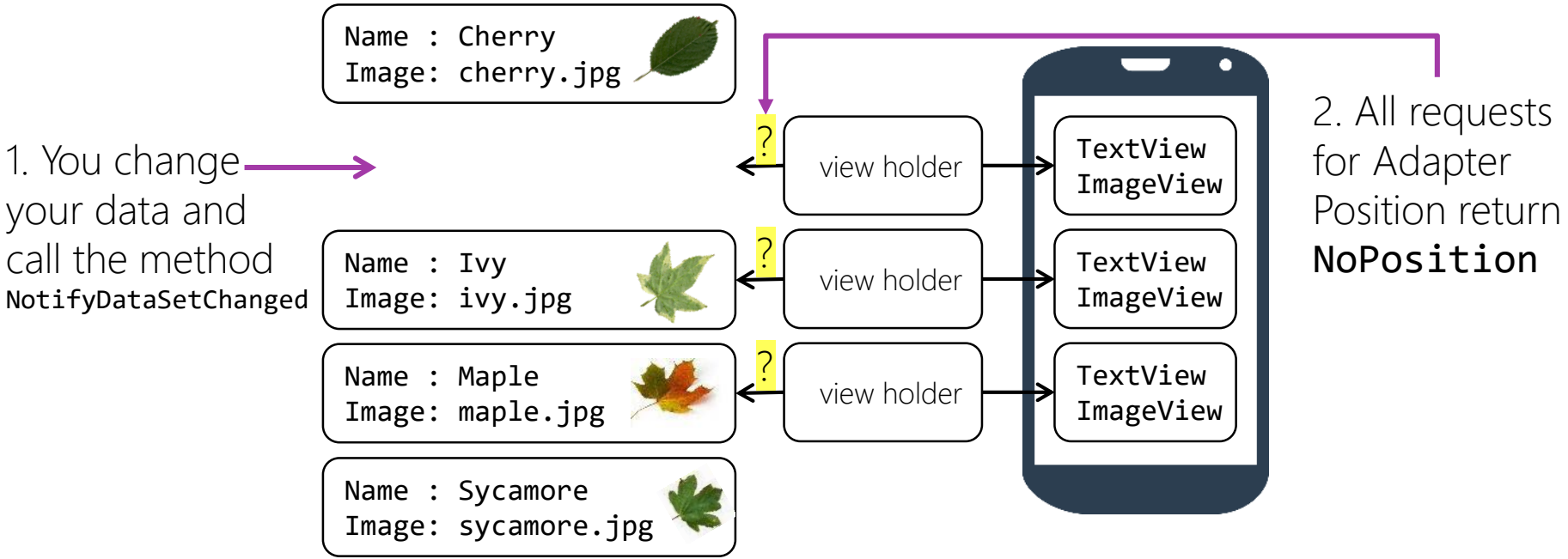
Adapter Position availability [removed]

- **`AdapterPosition`** returns **`RecyclerView.NoPosition`** if you ask for the position of a removed item before the next layout pass



Adapter Position availability [notify]

- **`AdapterPosition`** returns **`RecyclerView.NoPosition`** for all items after you call **`NotifyDataSetChanged`** until the next layout pass



When to use AdapterPosition

- Apps typically use **`AdapterPosition`** for item-click; it works well when you need to respond based on the data (e.g. master-details view)

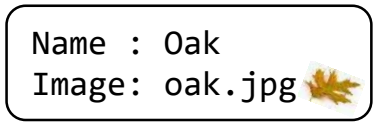
Get the position

Typical to discard events that yield **`NoPosition`**

```
public class MyViewHolder : RecyclerView.ViewHolder
{ ...
  void OnClick(object sender, EventArgs e)
  {
    int position = base.AdapterPosition;

    if (position == RecyclerView.NoPosition)
      return;

    ...
  }
}
```

View holder [responsibility]

- Your view holder is the natural place to detect user actions



It knows the item's position in your dataset

```
<LinearLayout ... >
  <TextView  ... />
  <ImageView ... />
</LinearLayout>
```

It has references to the entire layout and the views inside the layout

View holder [implementation]

- View holder should detect user clicks and report them to its adapter

```
public class MyViewHolder : RecyclerView.ViewHolder
{
  public MyViewHolder(View itemView, Action<int> listener)
    : base(itemView)
  {
    itemView.Click += (s, e) => listener(base.AdapterPosition);
    ...
  }
}
```

This example listens for clicks on the entire item layout, could also subscribe on the views inside if needed

Notify the adapter via a callback, pass the position of the clicked item (error checking for **NoPosition** omitted here)

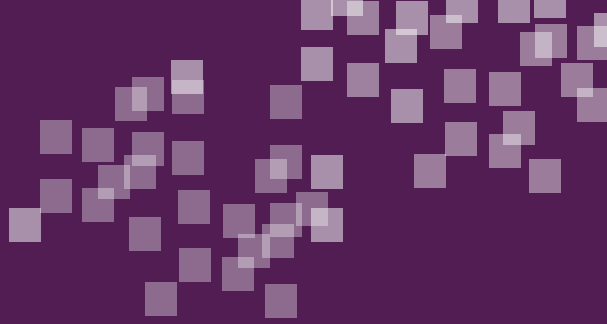
Adapter [implementation]

- Your adapter raises its event to notify client code

```
public class MyAdapter : RecyclerView.Adapter
{ ...
  public event EventHandler<int> ItemClick;

  public override RecyclerView.ViewHolder OnCreateViewHolder(ViewGroup parent, int viewType)
  { ...
    return new MyViewHolder(view, OnClick);
  }

  void OnClick(int position)
  {
    if (ItemClick != null)
      ItemClick(this, position);
  }
}
```

2. Raise the event

1. Register a callback with the view holder



Individual Exercise

Add an item-click event

Xamarin University

Summary

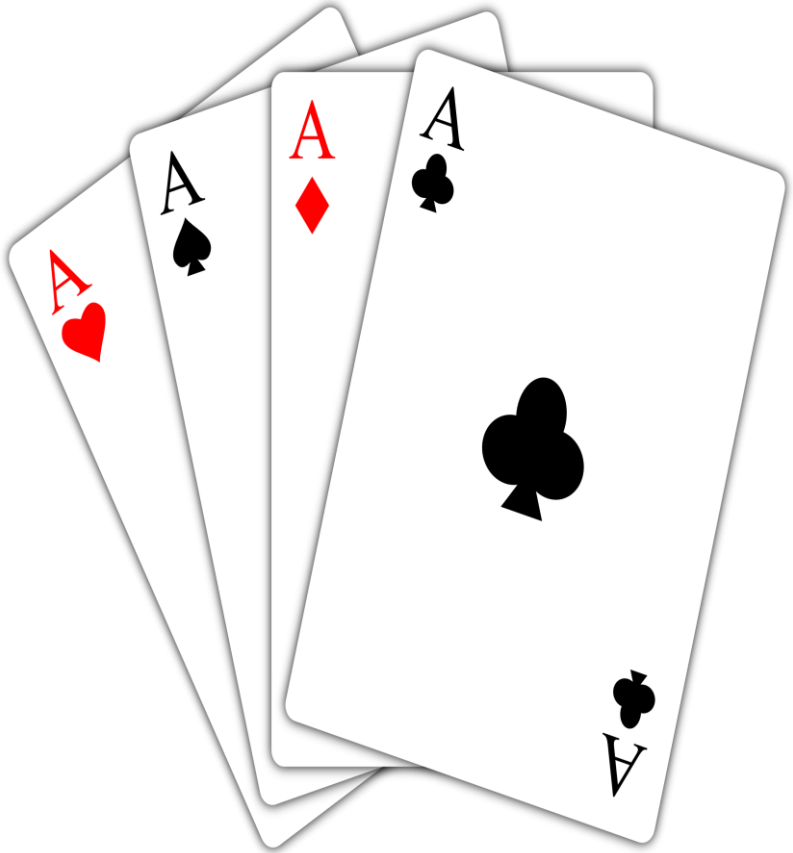
1. Determine the position of the clicked item
2. Detect user actions
3. Report user actions via an event

Show data in a CardView

Xamarin University

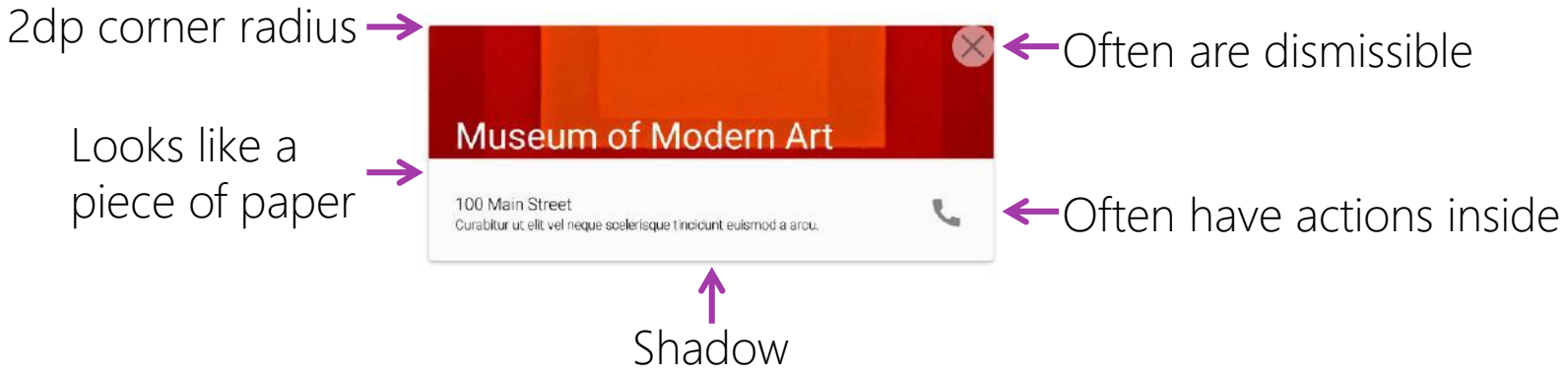
Tasks

1. Decide whether to use **`CardView`**
2. Add the **`CardView`** support library
3. Use **`CardView`** in your item-layout file



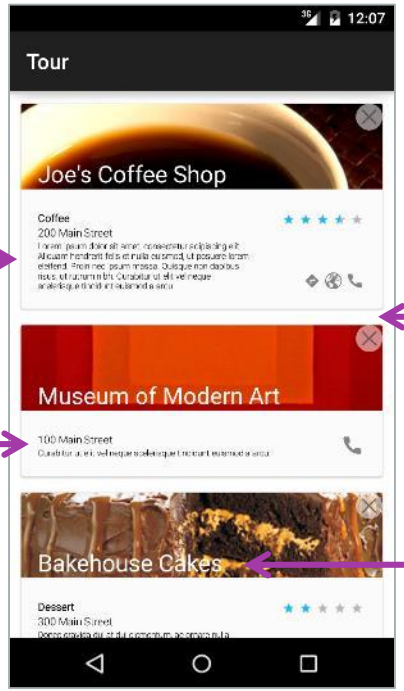
What is CardView?

- **`CardView`** is a container for displaying related data, typically used to display an item from a collection



CardView layout guidelines

- Android has guidelines for the content within a card and how to arrange card collections



8dp from edge of screen

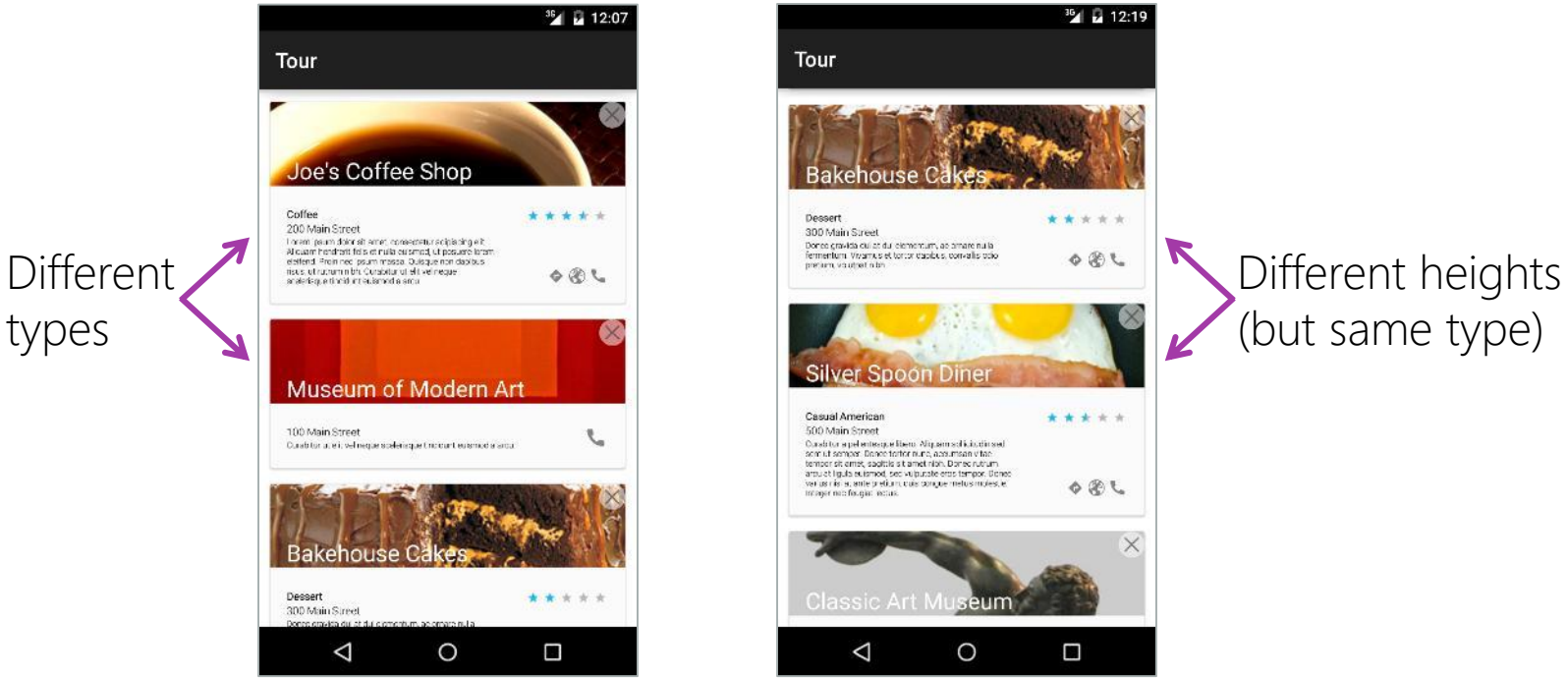
16dp content padding

8dp between cards

24sp min for headline text, 14sp or 16sp for body text

When to use CardView

- Use **`CardView`** for data that is variable type or variable size



CardView packaging

- **`CardView`** is also in a support library available from the Xamarin Component Store or Nuget

1. Add the Xamarin Component or the NuGet package

```
<android.support.v7.widget.CardView>
  ...
</android.support.v7.widget.CardView>
```

2. Qualify the name

`CardView` runs on older API levels but requires the app to be built with SDK level 21. Otherwise if will fail while inflating the layout

CardView attributes

- **`CardView`** offers several custom attributes that influence how it looks

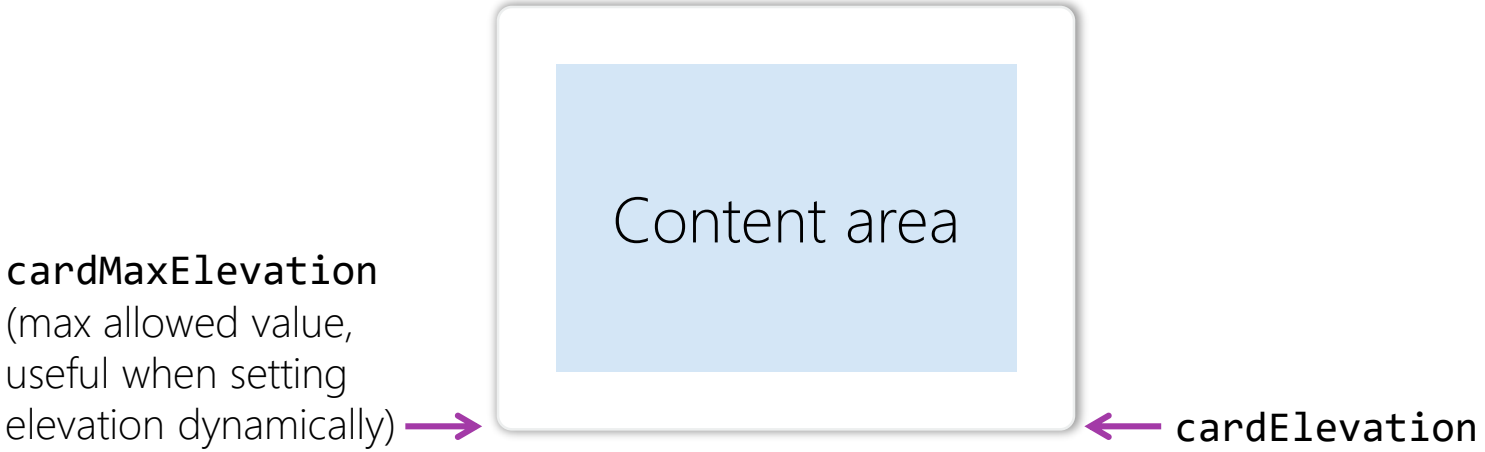
Declare namespace →

Set value →

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
  xmlns:custom="http://schemas.android.com/apk/res-auto"
  ...
  custom:cardElevation="5dp">
  ...
</android.support.v7.widget.CardView>
```

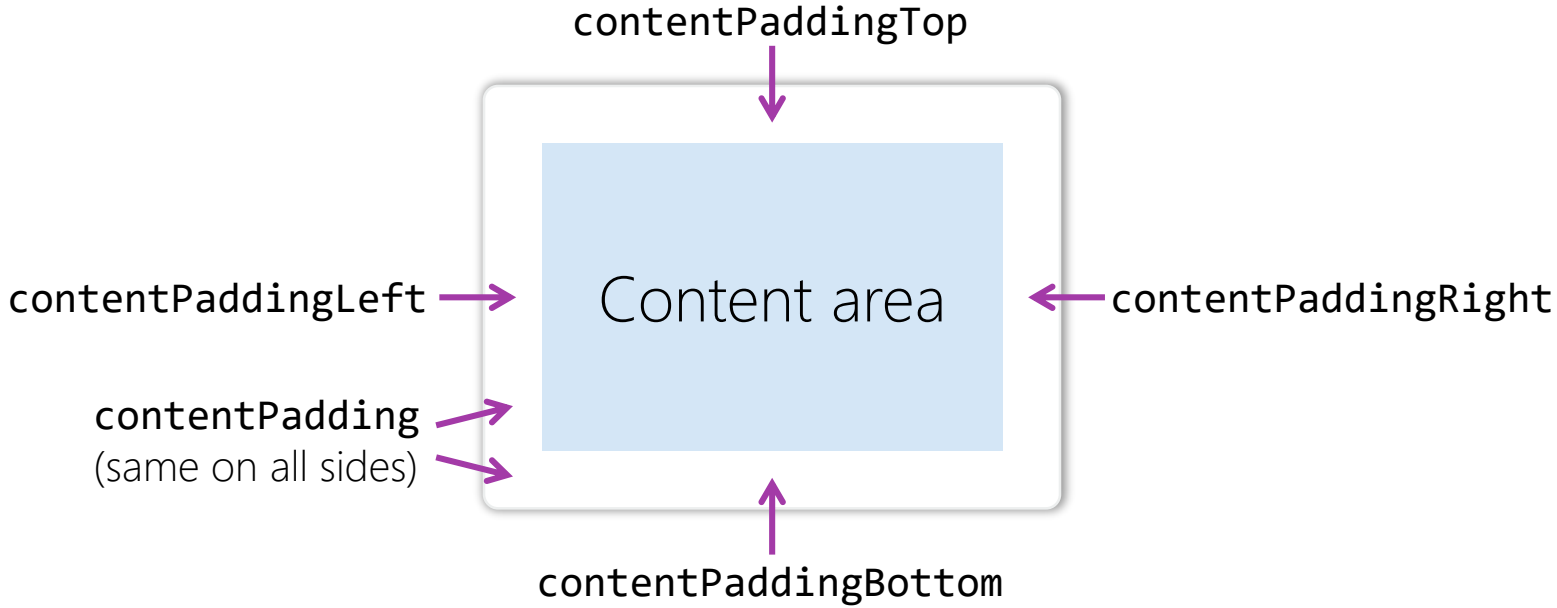
CardView elevation

- **`CardView`** lets you control elevation (i.e. shadow size) to make the card appear to float, larger values make it look higher



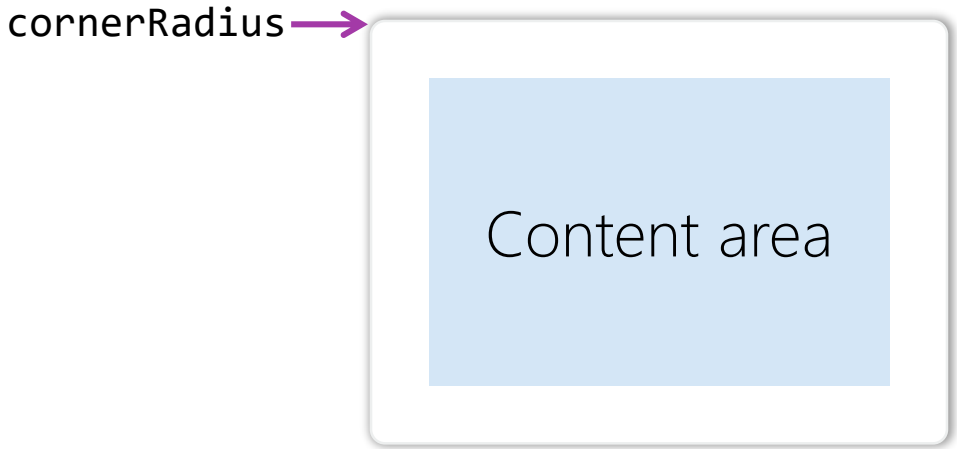
CardView padding

- **`CardView`** offers 5 padding properties to inset content within the card; they can be set via XML (see below) or via analogous methods in code



CardView corner radius

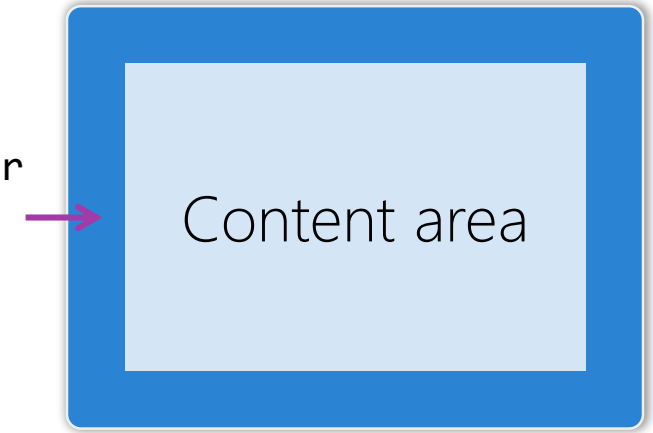
- **`CardView`** lets you control the corner radius, the recommended value is 2dp (0dp is discouraged since it will look like a tile instead of a card)



CardView background

- **`CardView`** lets you control the background color

`cardBackgroundColor`
(e.g. the color shown here is **`#2A84D3`**)

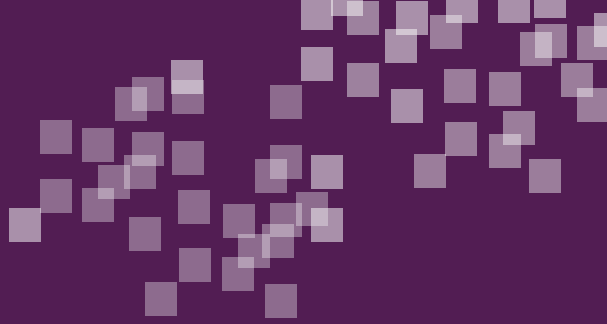


How to use CardView

- **`CardView`** is a **`FrameLayout`** that displays a single piece of your content

Typical to nest a layout inside a card

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView ... >

  <LinearLayout ... >
    ...
  </LinearLayout ... >

</android.support.v7.widget.CardView>
```



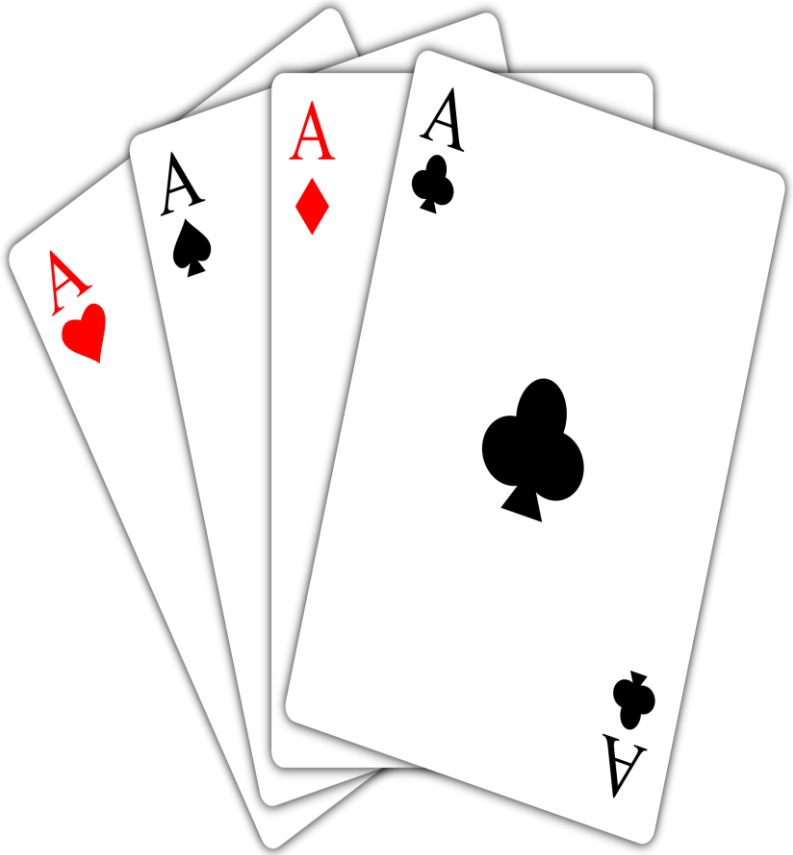
Group Exercise

Show data in a CardView

Xamarin University

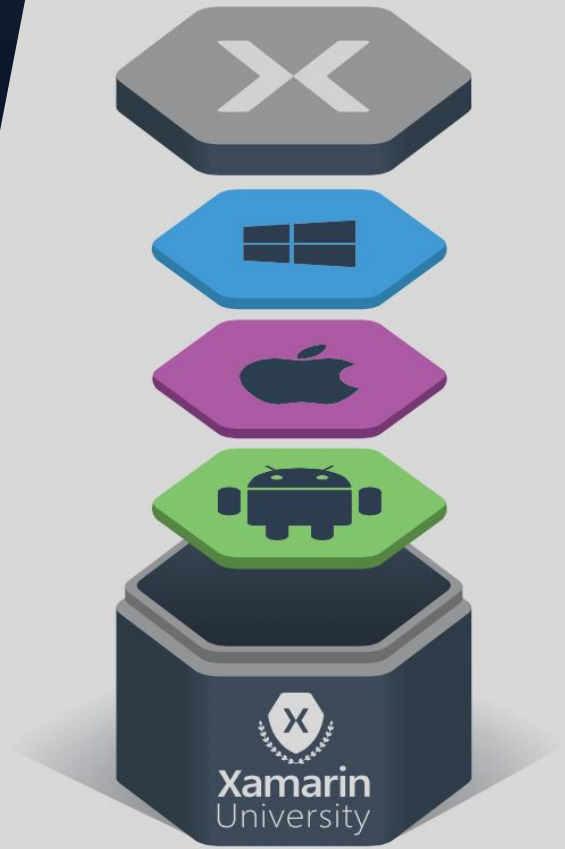
Summary

1. Decide whether to use **`CardView`**
2. Add the **`CardView`** support library
3. Use **`CardView`** in your item-layout file



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile



Microsoft