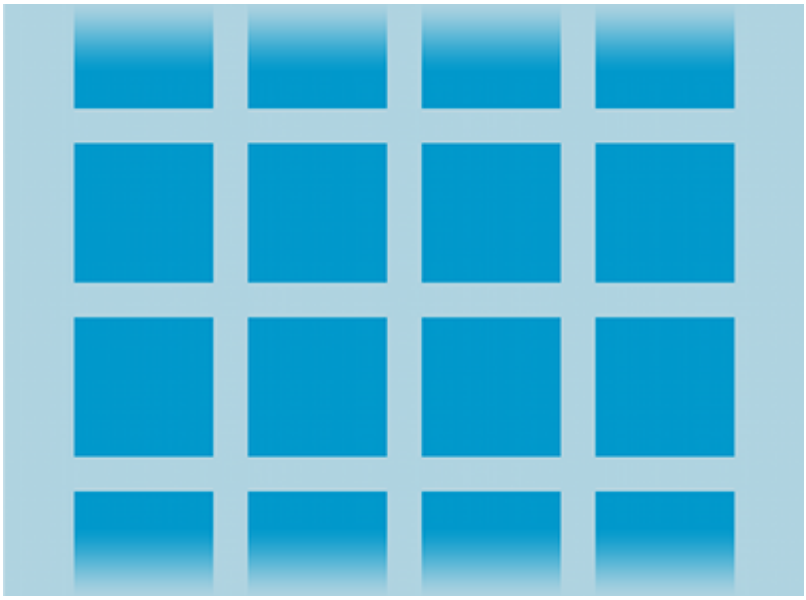
Grid View

`GridView` (https://developer.android.com/reference/android/widget/GridView.html) is a `ViewGroup` (https://developer.android.com/reference/android/view/ViewGroup.html) that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a `ListAdapter` (https://developer.android.com/reference/android/widget/ListAdapter.html).

Note: For better performance and tooling support, you should instead build your layout with ConstraintLayout (https://developer.android.com/training/constraint-layout/index.html).

For an introduction to how you can dynamically insert views using an adapter, read Building Layouts with an Adapter (https://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews).



Example

In this tutorial, you'll create a grid of image thumbnails. When an item is selected, a toast message will display the position of the image.

1. Start a new project named *HelloGridView*.
2. Find some photos you'd like to use, or download these sample images (https://developer.android.com/shareables/sample_images.zip). Save the image files into the project's `res/drawable/` directory.

3. Open the **`res/layout/main.xml`** file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

This **`GridView`** (https://developer.android.com/reference/android/widget/GridView.html) will fill the entire screen. The attributes are rather self explanatory. For more information about valid attributes, see the **`GridView`** (https://developer.android.com/reference/android/widget/GridView.html) reference.

4. Open **`HelloGridView.java`** and insert the following code for the **`onCreate()`** (https://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)) method:

KOTLIN JAVA

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v,
                int position, long id) {
            Toast.makeText(HelloGridView.this, "" + position,
                    Toast.LENGTH_SHORT).show();
        }
    });
}
```

After the **`main.xml`** layout is set for the content view, the **`GridView`** (https://developer.android.com/reference/android/widget/GridView.html) is captured from the layout with **`findViewById(int)`**

(https://developer.android.com/reference/android/app/Activity.html#findViewById(int)). The **`setAdapter()`** (https://developer.android.com/reference/android/widget/AdapterView.html#setAdapter(T)) method then sets a custom adapter (**`ImageAdapter`**) as the source for all items to be displayed in the grid. The **`ImageAdapter`** is created in the next step.

To do something when an item in the grid is clicked, the **`setOnItemClickListener()`** (https://developer.android.com/reference/android/widget/AdapterView.html#setOnItemClickListener(android.widget.AdapterView.OnItemClickListener)) method is passed a new **`AdapterView.OnItemClickListener`** (https://developer.android.com/reference/android/widget/AdapterView.OnItemClickListener.html). This anonymous instance defines the **`onItemClick()`** (https://developer.android.com/reference/android/widget/AdapterView.OnItemClickListener.html#onItemClick(android.widget.AdapterView<?>, android.view.View, int, long)) callback method to show a **`Toast`** (https://developer.android.com/reference/android/widget/Toast.html) that displays the index position (zero-based) of the selected item (in a real world scenario, the position could be used to get the full sized image for some other task).

5. Create a new class called **`ImageAdapter`** that extends **`BaseAdapter`** (https://developer.android.com/reference/android/widget/BaseAdapter.html):

KOTLIN JAVA

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent
```

```
        ImageView imageView;
        if (convertView == null) {
            // if it's not recycled, initialize some attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new ViewGroup.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }

    // references to our images
    private Integer[] mThumbIds = {
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7
    };
}
```

First, this implements some required methods inherited from **`BaseAdapter`** (https://developer.android.com/reference/android/widget/BaseAdapter.html). The constructor and **`getCount()`** (https://developer.android.com/reference/android/widget/Adapter.html#getCount()) are self-explanatory. Normally, **`getItem(int)`** (https://developer.android.com/reference/android/widget/Adapter.html#getItem(int)) should return the actual object at the specified position in the adapter, but it's ignored for this example. Likewise, **`getItemId(int)`** (https://developer.android.com/reference/android/widget/Adapter.html#getItemId(int)) should return the row id of the item, but it's not needed here.

The first method necessary is **`getView()`** (https://developer.android.com/reference/android/widget/Adapter.html#getView(int, android.view.View, android.view.ViewGroup)) . This method creates a new **`View`**

(https://developer.android.com/reference/android/view/View.html) for each image added to the `ImageAdapter`. When this is called, a `View` (https://developer.android.com/reference/android/view/View.html) is passed in, which is normally a recycled object (at least after this has been called once), so there's a check to see if the object is null. If it *is* null, an `ImageView` (https://developer.android.com/reference/android/widget/ImageView.html) is instantiated and configured with desired properties for the image presentation:

- `setLayoutParams(ViewGroup.LayoutParams)` (https://developer.android.com/reference/android/view/View.html#setLayoutParams(android.view.ViewGroup.LayoutParams))
  sets the height and width for the View—this ensures that, no matter the size of the drawable, each image is resized and cropped to fit in these dimensions, as appropriate.
- `setScaleType(ImageView.ScaleType)` (https://developer.android.com/reference/android/widget/ImageView.html#setScaleType(android.widget.ImageView.ScaleType))
  declares that images should be cropped toward the center (if necessary).
- `setPadding(int, int, int, int)` (https://developer.android.com/reference/android/view/View.html#setPadding(int, int, int, int))
  defines the padding for all sides. (Note that, if the images have different aspect-ratios, then less padding will cause more cropping of the image if it does not match the dimensions given to the ImageView.)

If the `View` (https://developer.android.com/reference/android/view/View.html) passed to `getView()` (https://developer.android.com/reference/android/widget/Adapter.html#getView(int, android.view.View, android.view.ViewGroup)) is *not* null, then the local `ImageView` (https://developer.android.com/reference/android/widget/ImageView.html) is initialized with the recycled `View` (https://developer.android.com/reference/android/view/View.html) object.

At the end of the `getView()` (https://developer.android.com/reference/android/widget/Adapter.html#getView(int, android.view.View, android.view.ViewGroup)) method, the `position` integer passed into the method is used to select an image from the `mThumbIds` array, which is set as the image resource for the `ImageView` (https://developer.android.com/reference/android/widget/ImageView.html).

All that's left is to define the `mThumbIds` array of drawable resources.

6. Run the application.

Try experimenting with the behaviors of the `GridView` (https://developer.android.com/reference/android/widget/GridView.html) and `ImageView` (https://developer.android.com/reference/android/widget/ImageView.html) elements by adjusting their properties. For example, instead of using `setLayoutParams(ViewGroup.LayoutParams)` (https://developer.android.com/reference/android/view/View.html#setLayoutParams(android.view.ViewGroup.LayoutParams)), try using `setAdjustViewBounds(boolean)` (https://developer.android.com/reference/android/widget/ImageView.html#setAdjustViewBounds(boolean)).

Last updated April 17, 2018.

Twitter
Follow @AndroidDev on Twitter

Google+
Follow Android Developers on Google+

YouTube
Check out Android Developers on YouTube