
YALOVA UNIVERSITY

HUAWEI

# Android Programming with Huawei Mobile Services

Yunus OZEN
Muhammed Salih KARAKASLI
Cenk Faruk CAVGA
Sezer Yavuzer BOZKIR


# Android Basics

## Core Topics

- Activities
- Fragments
- Data & File Storage

## UI & Navigation

- Layouts
- Adapters
- Image & Graphics
- Animations

## Device Compabilities

- Support Different Screen Size
- Support Different Languages

## Example Project

- Let's try what we learn today!

# Core Topics

# Activities

An activity provides the **window** in which the app **draws its UI.**
This window typically fills the screen, but may be smaller than the screen.
Generally, one activity implements one screen in an app.

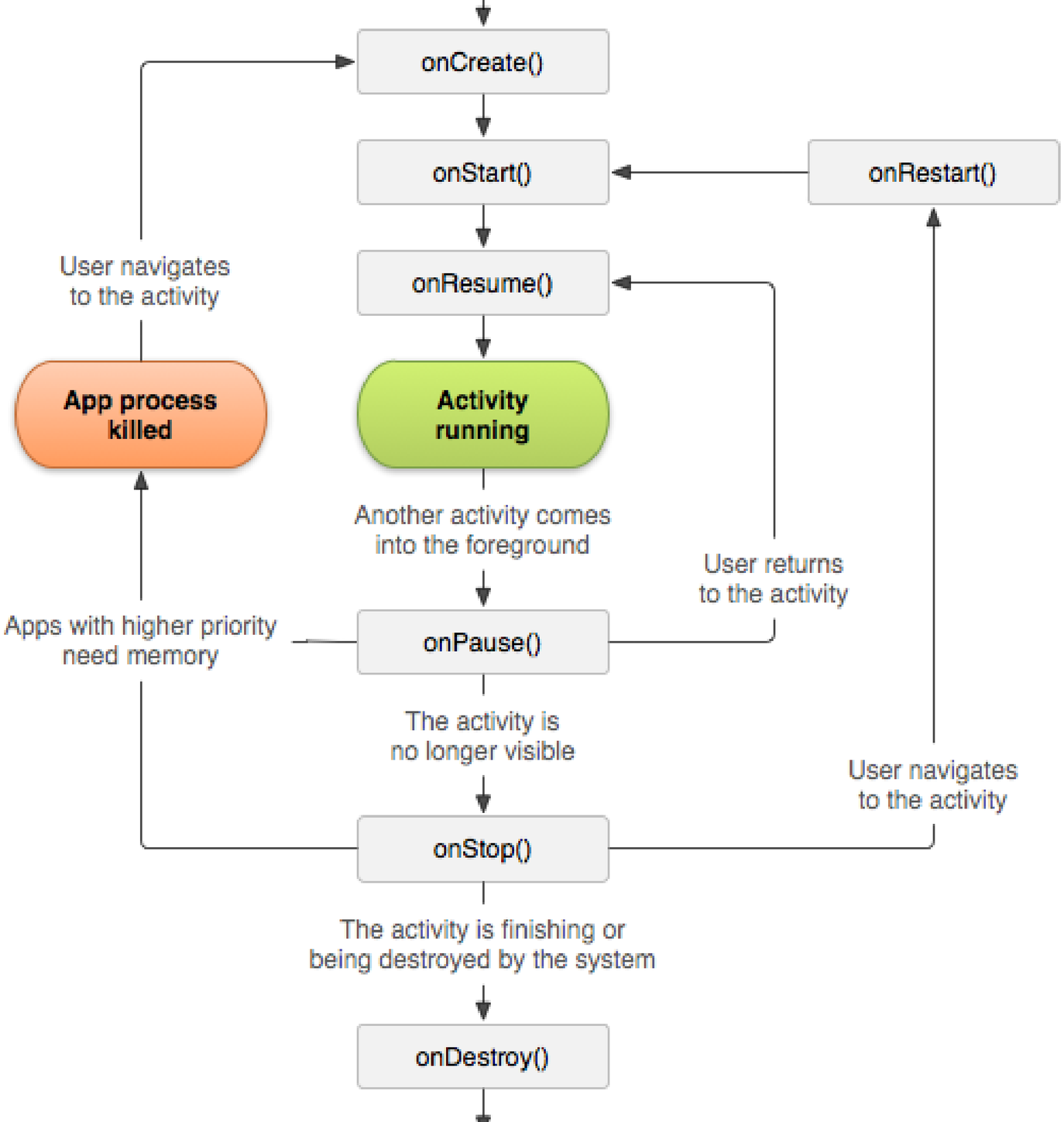
Most apps **contain multiple screens**, which means they comprise **multiple activities**.

Each activity can then **start another activity** in order to perform different actions.

# Understand Activity Life Cycle

The Activity class **provides a number of callbacks** that allow the activity to know that a state has changed: that the system is **creating**, **stopping**, or **resuming** an activity, or **destroying** the process in which the activity resides.

Within the lifecycle callback methods, you can declare **how your activity behaves** when the user leaves and re-enters the activity.



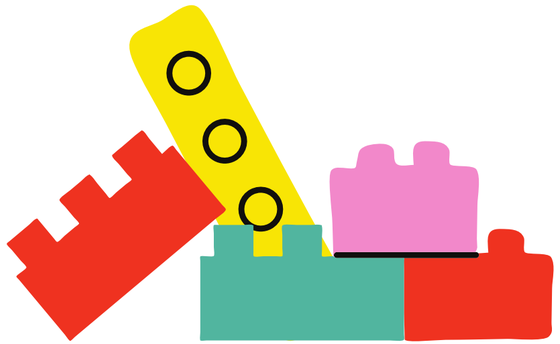

# Activity Life Cycle

To navigate transitions between stages of the activity lifecycle, the Activity class provides a core **set of six callbacks**:

1. onCreate(),
2. onStart(),
3. onResume(),
4. onPause(),
5. onStop()
6. onDestroy()

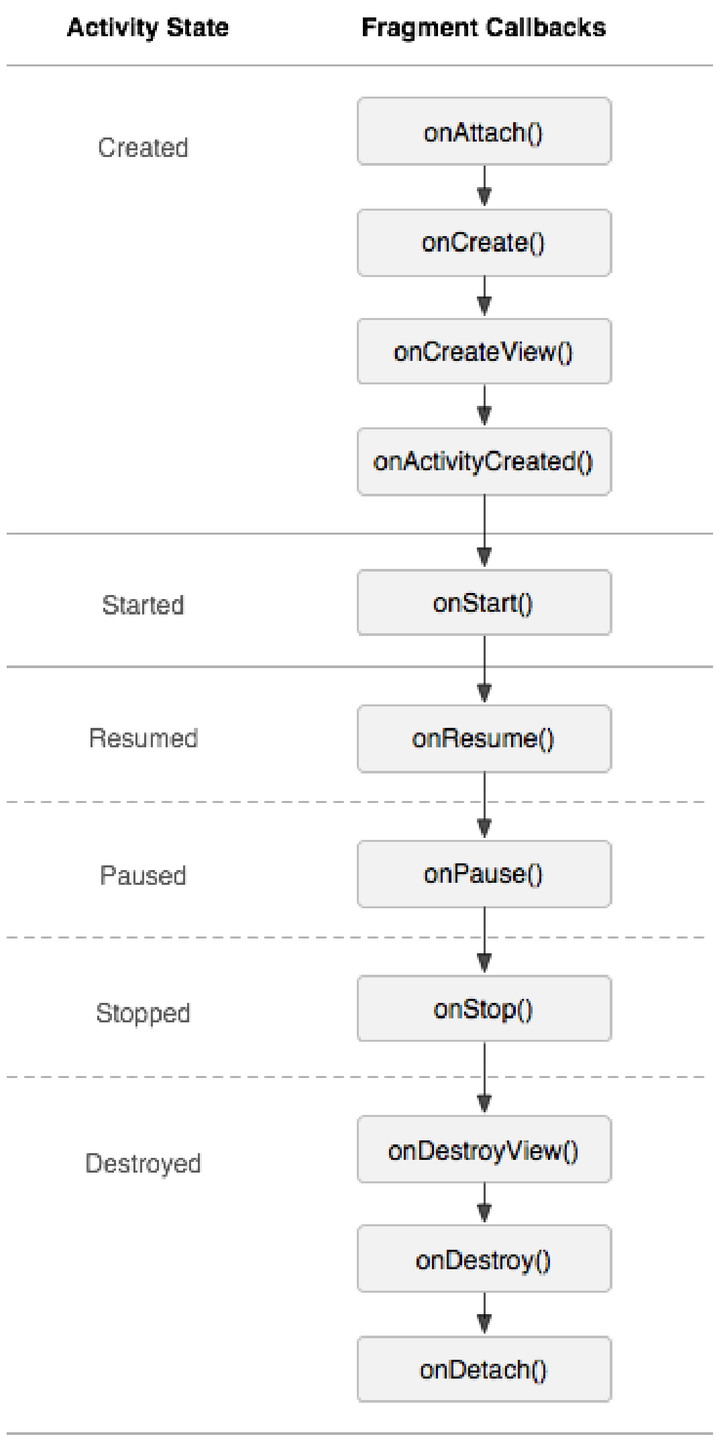
# Fragments

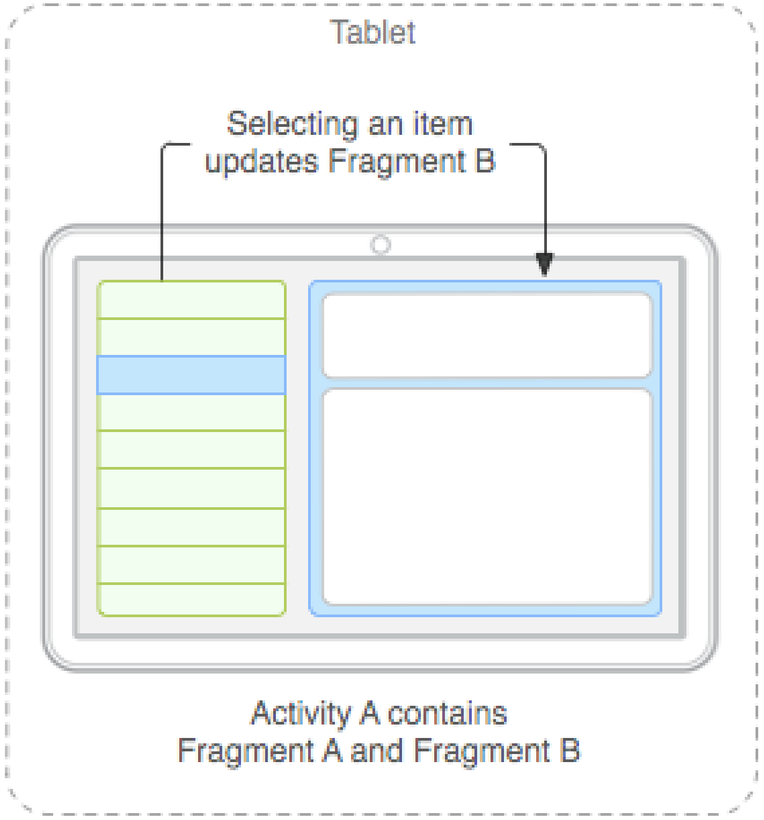
A Fragment represents a behavior or a **portion of user interface** in a FragmentActivity. You can combine **multiple fragments in a single activity** to build a multi-pane UI and **reuse a fragment** in multiple activities.

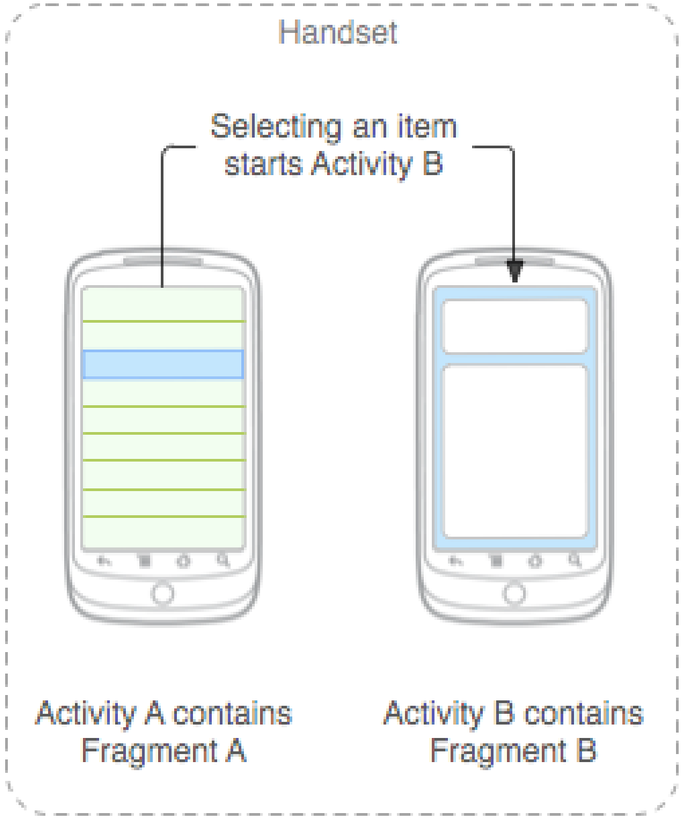


A fragment **must always be hosted in an activity** and the fragment's lifecycle is directly affected by the host activity's lifecycle.

You should design each fragment as a **modular and reusable** activity component.


Activity State
Fragment Callbacks
Created
onAttach()
onCreate()
onCreateView()
onActivityCreated()
Started
onStart()
Resumed
onResume()
Paused
onPause()
Stopped
onStop()
Destroyed
onDestroyView()
onDestroy()
onDetach()


Tablet
Selecting an item
updates Fragment B
Activity A contains
Fragment A and Fragment B


Handset
Selecting an item
starts Activity B
Activity A contains
Fragment A
Activity B contains
Fragment B

# Data & File Storage

Android uses a file system that's similar to disk-based file systems on other platforms. The system provides several options for you to save your app data:

1. **App-specific storage:** Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.
2. **Shared storage:** Store files that your app intends to share with other apps, including media, documents, and other files.
3. **Preferences:** Store private, primitive data in key-value pairs.
4. **Databases:** Store structured data in a private database using the Room persistence library.

# UI & Navigation

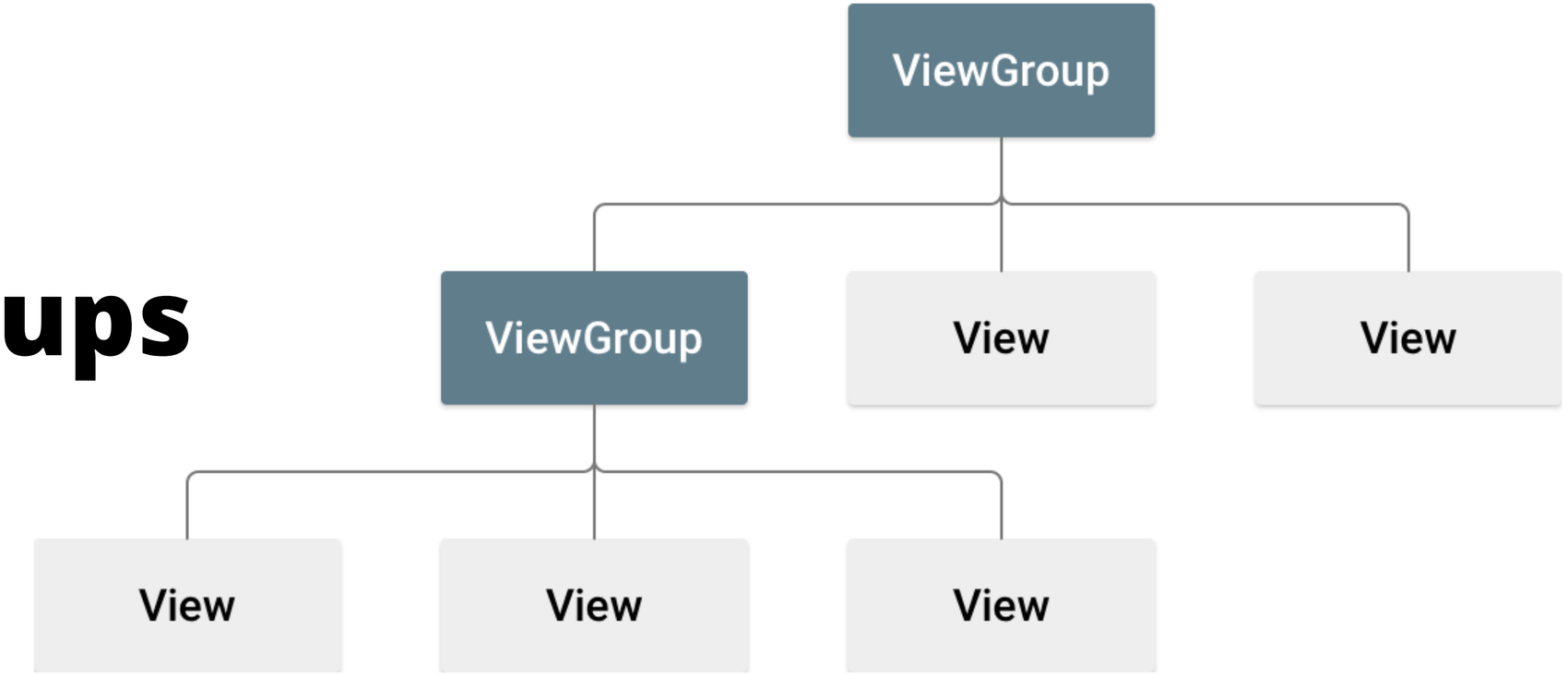
# User Interface

Your app's user interface is everything that the user can see and interact with.

Android provides a variety of **pre-built UI components** such as structured layout objects and **UI controls** that allow you to build the graphical user interface for your app.

Android also provides other UI modules for special interfaces such as **dialogs**, **notifications**, and **menus**.

# Views & View Groups




A layout defines the **structure for a user interface** in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. **A View usually draws something** the user can see and interact with. Whereas a **ViewGroup is an invisible container** that defines the layout structure for View and other ViewGroup objects, as shown in figure 1.

# Example

Let's discuss view groups and views!

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

# View Attributes

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

- Any View object may have an integer **ID** associated with it, to uniquely identify the View within the tree.
- All view groups include a width and height (**layout_width and layout_height**), and each view is required to define them.

    - **wrap_content** tells your view to size itself to the dimensions required by its content.
    - **match_parent** tells your view to become as big as its parent view group will allow.

# Simple Layouts

### Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

### Relative Layout

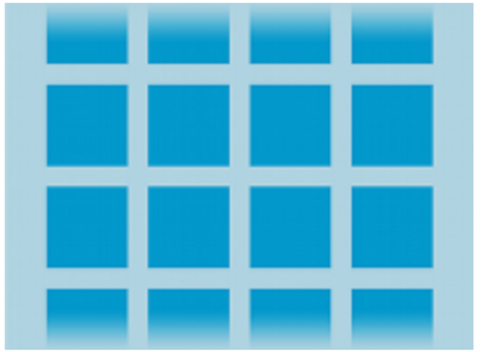
Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

# Layouts with Adapter

### List View

Displays a scrolling single column list.

### Grid View



Displays a scrolling grid of columns and rows.

# Adapters

- When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses AdapterView to populate the layout with views at runtime.
- Adapter to bind data to its layout.
- The Adapter behaves as a middleman between the data source and the AdapterView layout.
- The Adapter retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the AdapterView layout.

## Adapter Types

**Array Adapter:** By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a TextView.

**SimpleCursorAdapter:** Use this adapter when your data comes from a Cursor. When using SimpleCursorAdapter, you must specify a layout to use for each row in the Cursor and which columns in the Cursor should be inserted into which views of the layout.

# Lets Try Array Adapter

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    String myList[] = new String[] {"İstanbul", "Ankara", "Sivas", "Niğde", "Muş",
            "Adana", "İzmir", "Kayseri", "Urfa", "Antep", "Sakarya", "Muğla", "Aydın",
            "Kırşehir", "Edirne", "Eskişehir", "Ağrı", "Kars", "Yalova"};

    ArrayAdapter<String> myArrayAdapter = new ArrayAdapter<>( context: this, android.R.layout.simple_list_item_1, myList);

    ListView cityListView = findViewById(R.id.my_list_view);
    cityListView.setAdapter(myArrayAdapter);
}
```

# Images & Graphics

# Drawables

When you need to **display static images** in your app, you can **use the Drawable** class and its subclasses to draw shapes and images.

There are two ways to define and instantiate a Drawable besides using the class constructors:

- Inflate an i**mage resource** (a bitmap file) saved in your project.
- Inflate an **XML resource** that defines the drawable properties.

```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/my_image"
        android:contentDescription="@string/my_image_desc" />
```

```
// Instantiate an ImageView and define its properties
ImageView i = new ImageView(this);
i.setImageResource(R.drawable.my_image);
```

# Custom Drawable

```
public class MyDrawable extends Drawable {
    private final Paint redPaint;

    public MyDrawable() {
        // Set up color and text size
        redPaint = new Paint();
        redPaint.setARGB(255, 255, 0, 0);
    }

    @Override
    public void draw(Canvas canvas) {
        // Get the drawable's bounds
        int width = getBounds().width();
        int height = getBounds().height();
        float radius = Math.min(width, height) / 2;

        // Draw a red circle in the center
        canvas.drawCircle(width/2, height/2, radius, redPaint);
    }
```

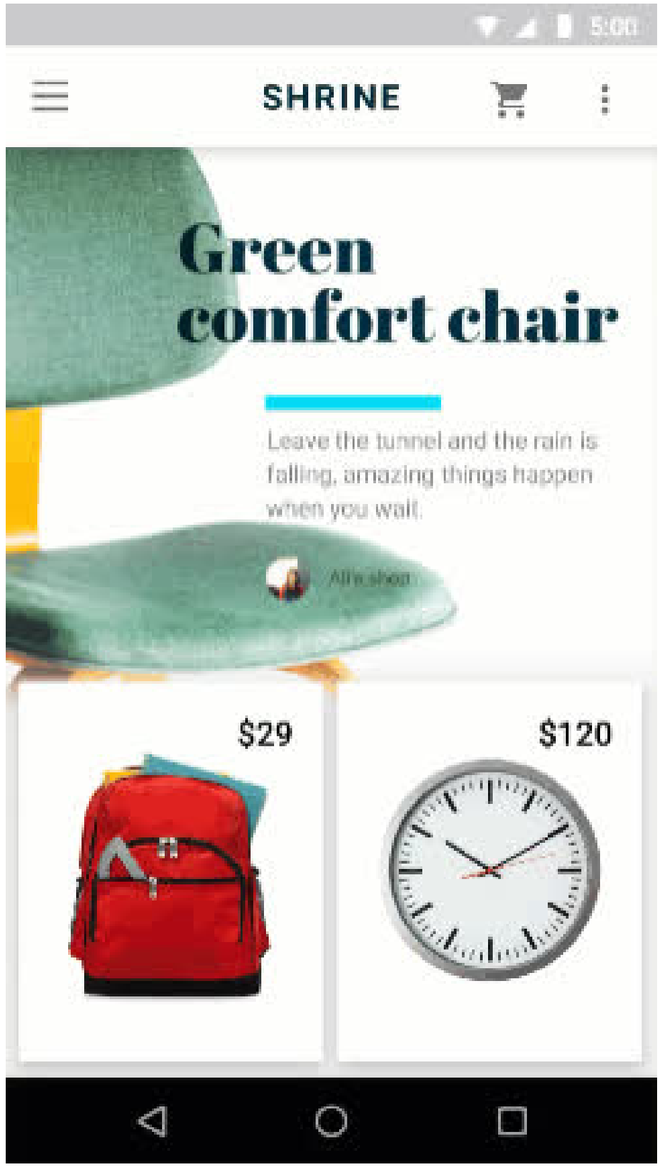
When you want to create some **custom drawings**, you can do so by **extending the Drawable class** (or any of its subclasses).The most important method to implement is draw(Canvas) because this provides the Canvas object you must use to provide your drawing instructions.

# Lets Try!

```
MyDrawable mydrawing = new MyDrawable();
ImageView image = findViewById(R.id.imageView);
image.setImageDrawable(mydrawing);
```

# Animations

- Animations can add visual cues that notify users about what's going on in your app.
- They are especially useful when the UI changes state, such as when new content loads or new actions become available.
- Animations also add a polished look to your app, which gives it a higher quality look and feel.



# Animations - Fade In/Out

```
// Animate the loading view to 0% opacity. After the animation ends,
// set its visibility to GONE as an optimization step (it won't
// participate in layout passes, etc.)
loadingView.animate()
        .alpha(0f)
        .setDuration(shortAnimationDuration)
        .setListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                loadingView.setVisibility(View.GONE);
            }
        });
```

# Animations - Move

```
ObjectAnimator animation = ObjectAnimator.ofFloat(view, "translationX", 100f);
animation.setDuration(2000);
animation.start();
```

# Device Compability

# Multiple Screen Size

Android devices come in all shapes and sizes, so your app's layout needs to be flexible. That is, instead of defining your layout with rigid dimensions that assume a certain screen size and aspect ratio, your layout should gracefully respond to different screen sizes and orientations.

**Avoid hard-coded layout sizes**

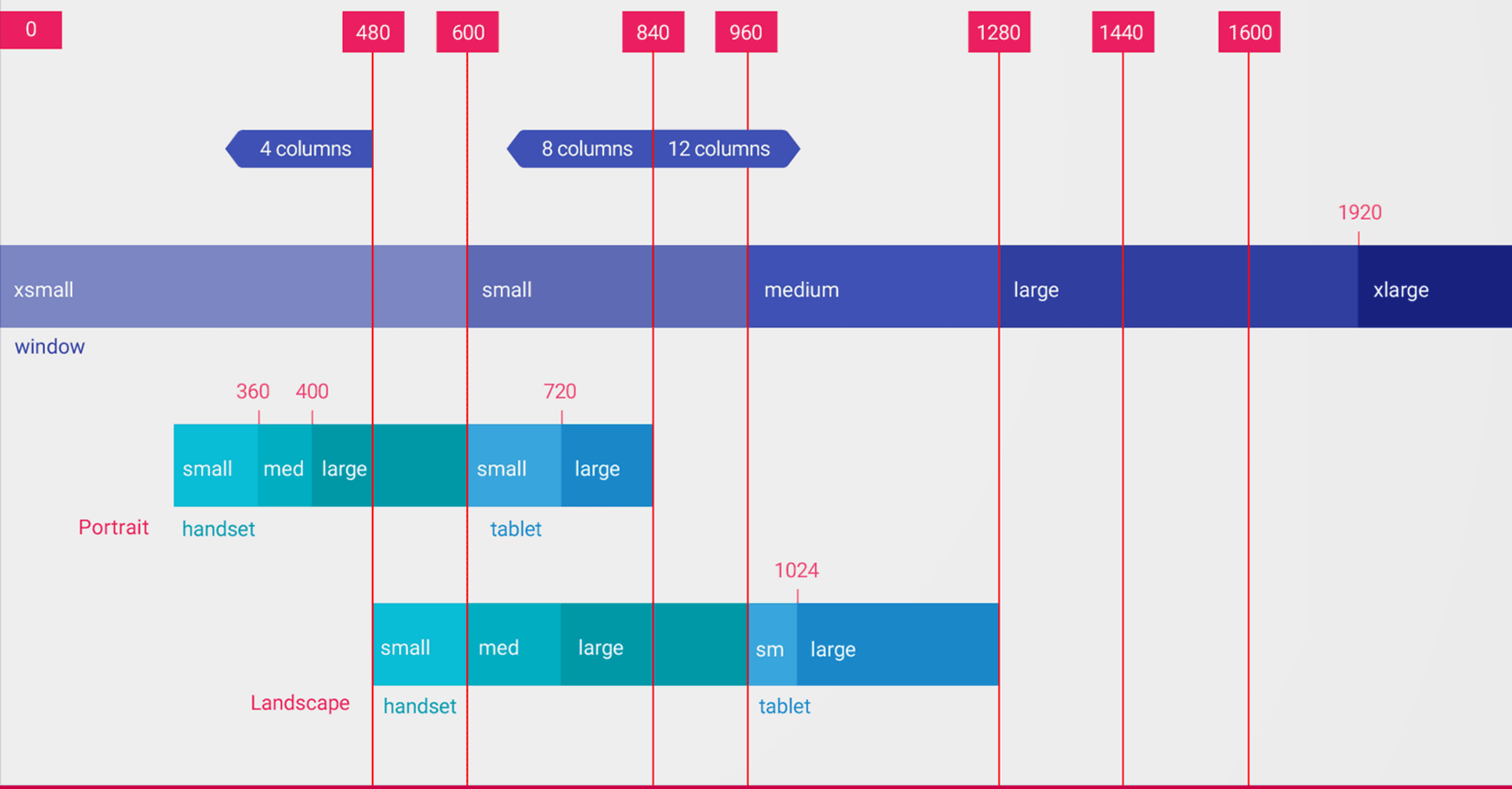
To ensure that your layout is flexible and adapts to different screen sizes, you should use "wrap_content" and "match_parent" for the width and height of most view components, instead of hard-coded sizes."wrap_content" tells the view to set its size to whatever is necessary to fit the content within that view."match_parent" makes the view expand to as much as possible within the parent view.

# Multiple Screen Size

**Use the smallest width qualifier**

The "smallest width" screen size qualifier allows you to provide alternative layouts for screens that have a minimum width measured in density-independent pixels(dp or dip).

```
res/layout/main_activity.xml           # For handsets (smaller than 600dp available width)
res/layout-sw600dp/main_activity.xml   # For 7” tablets (600dp wide and bigger)
```


0
480
600
840
960
1280
1440
1600
4 columns
8 columns
12 columns
1920
xsmall
small
medium
large
xlarge
window
360
400
720
small
med
large
small
large
Portrait
handset
tablet
1024
small
med
large
sm
large
Landscape
handset
tablet

# Multiple Screen Support

Android resolves language resources based on the system locale setting. You can provide support for different locales by using the resources directory in your Android project.

English strings (default locale), `/values/strings.xml`:

```
<resources>
    <string name="hello_world">Hello World!</string>
</resources>
```

Spanish strings (`es` locale), `/values-es/strings.xml`:

```
<resources>
    <string name="hello_world">¡Hola Mundo!</string>
</resources>
```

# Let's Try Something!

# LET'S TALK
## CONTACT INFORMATİON


Muhammed Salih KARAKASLI
muhammed.salih.karakasli@huawei.com

Cenk Faruk CAVGA
cenk.faruk.cavga@huawei.com

Sezer Yavuzer BOZKIR
sezer.bozkir@huawei.com


Telegram Channel
will be created

Official Website
https://developer.huawei.com