THE UNIVERSITY OF MELBOURNE

COMP90018

Mobile Computing Systems Programming

WEEK 2 – UI, Android Wear OS

Atiq Shaikh

atiq.shaikh@unimelb.edu.au

Intended Learning Outcomes

- Different Types of Layout
- Activity Lifecycle
- Android Wear OS

CONTENTS

- User Interface (UI)
- Android Wear OS

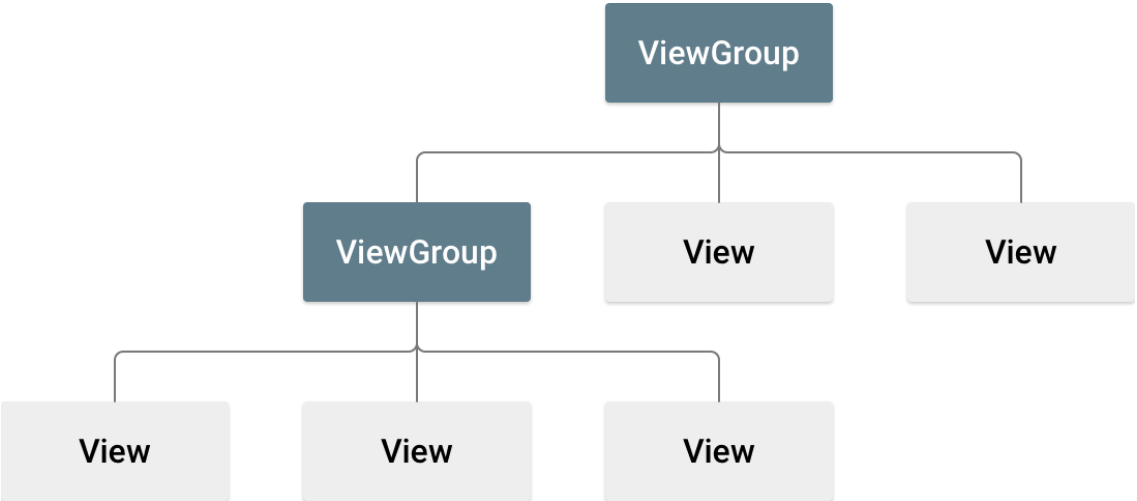
User Interface

User Interface

User interface in Android application is everything that the user can see and interact with.

https://developer.android.com/guide/topics/ui

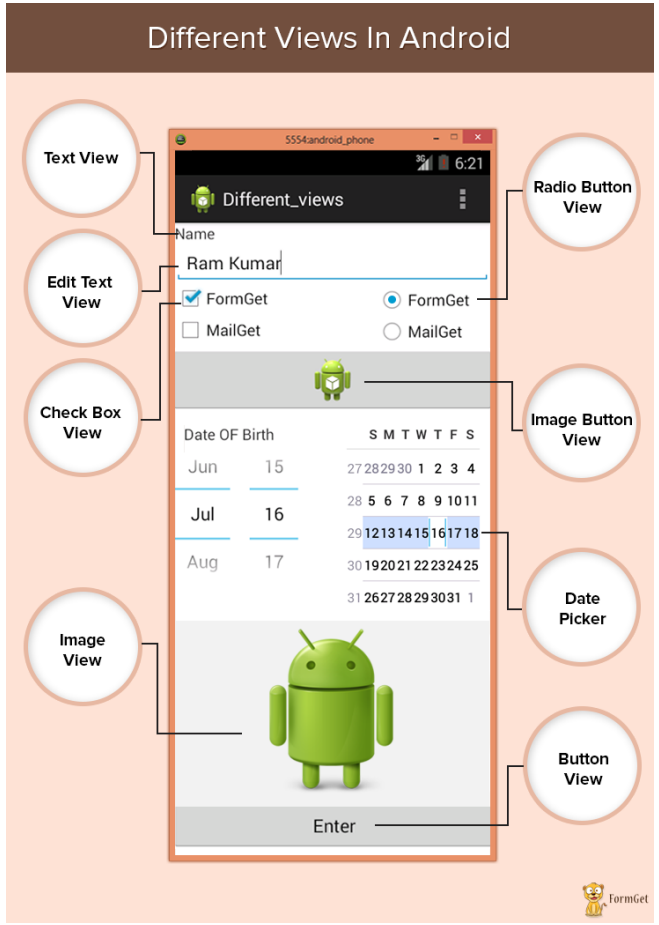
User Interface – Layout



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.

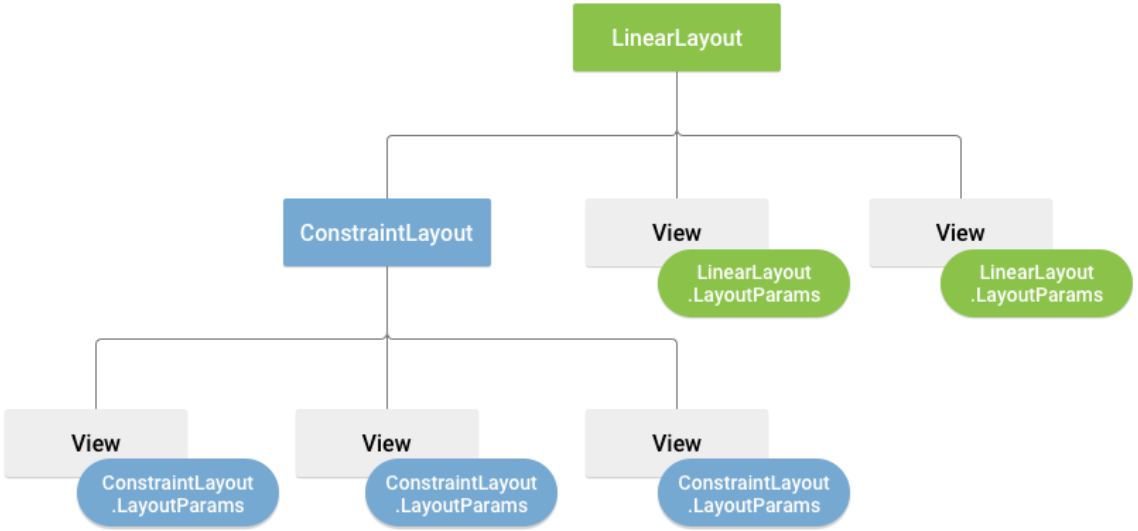
User Interface – View



A ***View*** usually draws something the *user can see and interact with*. The ***View*** objects are usually called "***widgets***" and can be one of many subclasses, such as *Button* or *TextView*.

https://developer.android.com/reference/android/view/View.html

User Interface – ViewGroup



A ***ViewGroup*** is an invisible container that defines the layout structure for ***View*** and other ***ViewGroup*** objects.

https://developer.android.com/reference/android/view/ViewGroup.html

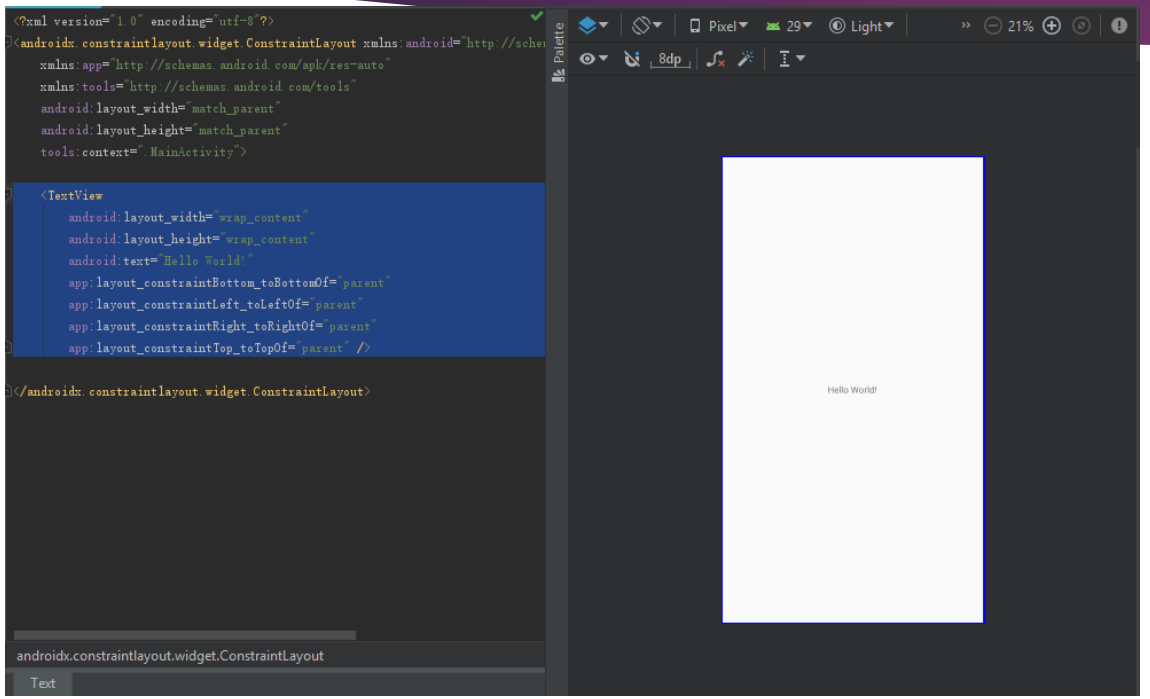
User Interface – ViewGroup

The ***ViewGroup*** objects are usually called "***layouts***" can be one of many types that provide a different layout ***structure***, such as *LinearLayout* or *ConstraintLayout* .

User Interface

- *Declare Layouts*

Declare a layout (XML)

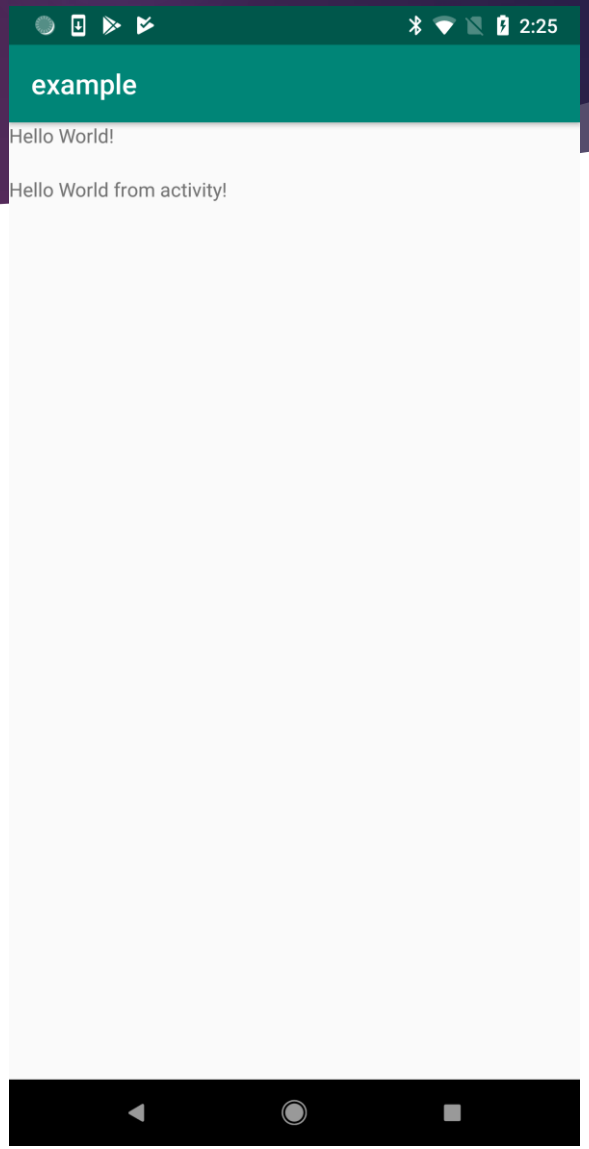


Declare UI elements in XML

Android provides a straightforward **XML** vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

Declare a layout (Runtime)

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

    LinearLayout layout = this.findViewById(R.id.first_activity);
    TextView helloWorld = new TextView( context: this);
    helloWorld.setLayoutParams(new ViewGroup.LayoutParams( width: -1, height: -2));
    helloWorld.setTextSize( unit: 1, size: 14);
    helloWorld.setText("Hello World from activity!");

    layout.addView(helloWorld);
}
```



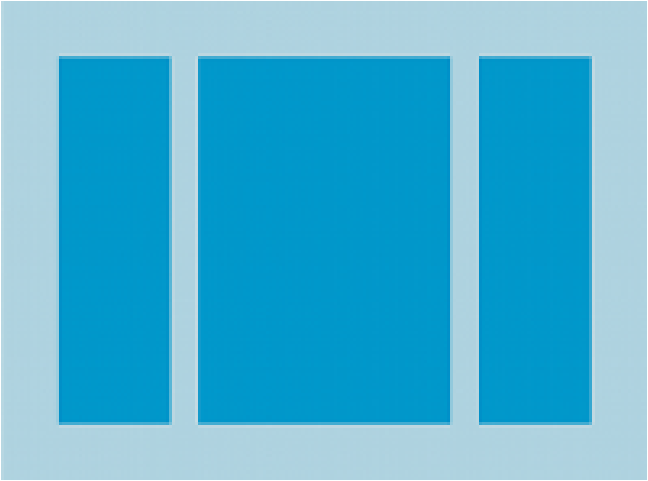
Instantiate layout elements at runtime

Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

User Interface

- *Explore Linear Layout and Relative Layout*

Linear Layout



Linear Layout

A layout that organizes its children into a **single *horizontal*** or ***vertical*** row.

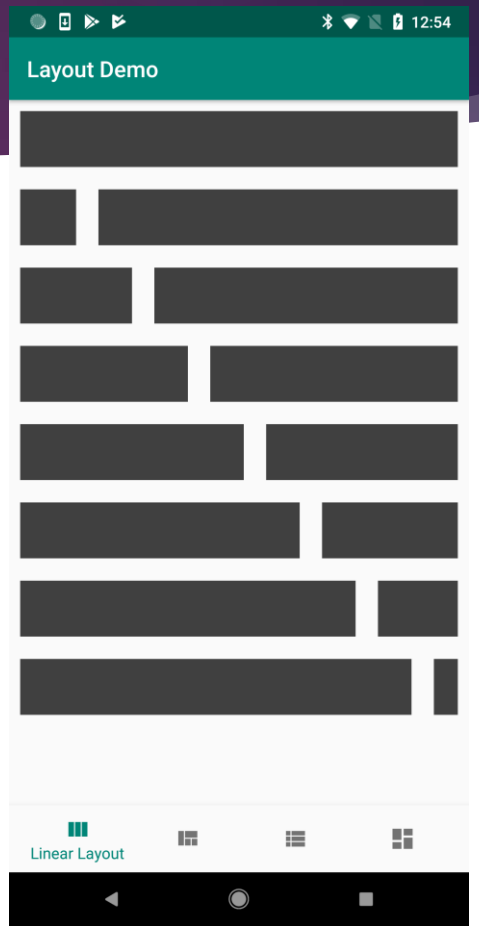
It creates a **scrollbar** *if the length of the window exceeds the length of the screen*.

https://developer.android.com/guide/topics/ui/layout/linear

Linear Layout (Code)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".LayoutDemoFragment">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:layout_margin="10dp"
            android:background="#BE000000" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_margin="10dp"
            android:background="#BE000000" />

        <TextView
            android:layout_width="match_parent"
            android:layout_height="50dp"
            android:layout_margin="10dp"
            android:background="#BE000000" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
```

fragment_layout_demo_linear.xml



Relative Layout

Relative Layout

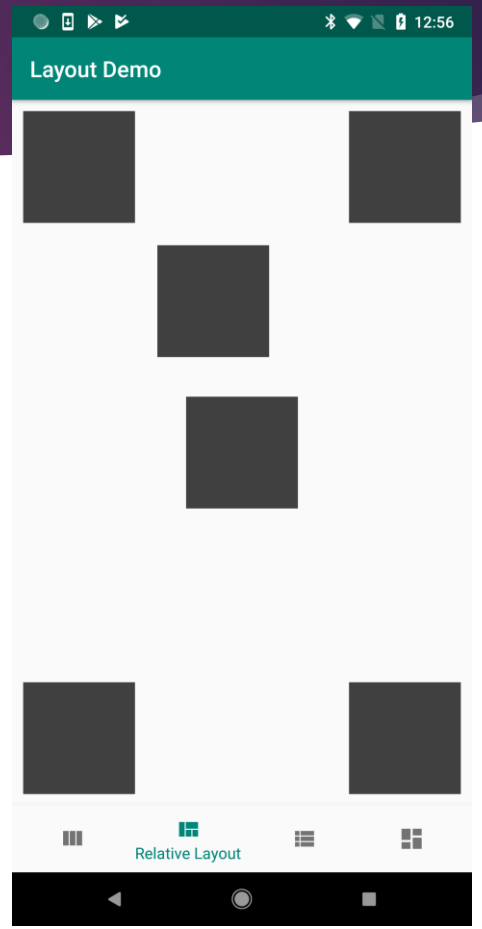
Enables you to ***specify*** the ***location of child objects*** relative to ***each other*** (child A to the left of child B) or to the ***parent*** (aligned to the top of the parent).

https://developer.android.com/guide/topics/ui/layout/relative

Relative Layout (Code)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".LayoutDemoFragment">

    <TextView
        android:id="@+id/corner_left_top"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_margin="10dp"
        android:background="#BE000000"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"/>

    <TextView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_margin="10dp"
        android:background="#BE000000"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"/>

    <TextView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_margin="10dp"
        android:background="#BE000000"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true"/>

    <TextView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_margin="10dp"
        android:background="#BE000000"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true"/>

    <TextView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_margin="10dp"
```

fragment_layout_demo_relative.xml

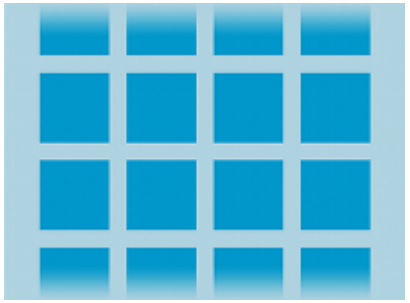


User Interface

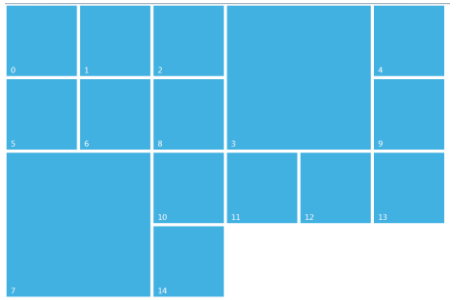
- Explore Layouts require an Adapter

Adapter

List View



Grid View



Recycler View

.....

What is adapter?

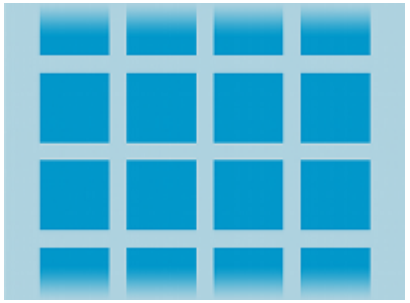
The ***Adapter*** behaves as a ***middleman*** between the *data source* and the *AdapterView layout*:

- ***Retrieves the data*** (from a source such as an array or a database query)

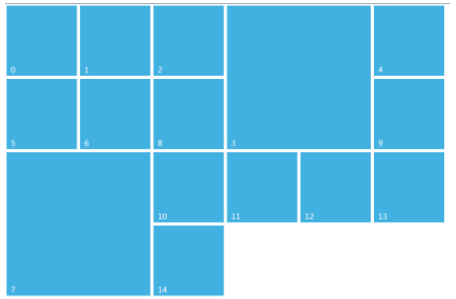
- ***Converts each entry into a view*** that can be added into the AdapterView layout.

Adapter

List View



Grid View



Recycler View

.....

Why do we need an adapter?

When the content for the layout is ***dynamic or not pre-determined***, you can use a layout that subclasses ***AdapterView*** to *populate the layout with views at runtime*.

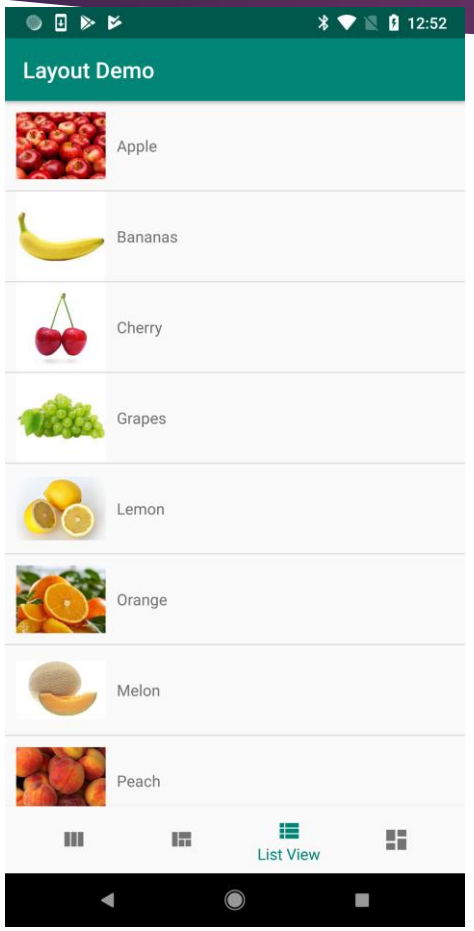
A subclass of the ***AdapterView*** class uses an ***Adapter*** to *bind data to its layout*.

User Interface

- *List View*

(Displays a scrolling single column list)

List View



To create a ***List View***:

- ***Declare*** *ListView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with ListView in the Activity*

https://developer.android.com/reference/android/widget/ListView

List View

fragment_layout_demo_list.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".LayoutDemoFragment">

    <ListView
        android:id="@+id/demo_list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

fragment_layout_demo_list.xml

- ***Declare*** *ListView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with ListView in the Activity*

List View

```java
package com.example.layoutdemo;


// Fruit class contains fruitImage and fruitName
// With automatically generated Setter and Getter
public class Fruit {
    // fruitImage to store the resource id if fruit image
    private int fruitImage;
    // fruitName to store the string of fruit name
    private String fruitName;

    public Fruit(int fruitImage, String fruitName) {
        this.fruitImage = fruitImage;
        this.fruitName = fruitName;
    }

    public int getFruitImage() { return fruitImage; }

    public void setFruitImage(int fruitImage) { this.fruitImage = fruitImage; }

    public String getFruitName() { return fruitName; }

    public void setFruitName(String fruitName) { this.fruitName = fruitName; }
}
```

Fruit.java

- ***Declare*** *ListView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with ListView in the Activity*

List View

```java
public class ListDemoAdapter extends ArrayAdapter<Fruit> {
    static class ViewHolder {
        @BindView(R.id.list_example_image)
        ImageView image;
        @BindView(R.id.list_example_text)
        TextView text;

        ViewHolder(View view) { ButterKnife.bind( target: this, view); }
    }

    private int resourceId;

    public ListDemoAdapter(Context context, int resource, List<Fruit> objects) {
        super(context, resource, objects);
        resourceId = resource;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Fruit fruit = getItem(position);
        View view;
        ViewHolder viewHolder;
        if (convertView == null) {
            view = LayoutInflater.from(getContext()).inflate(resourceId, root: null);
            viewHolder = new ViewHolder(view);
            view.setTag(viewHolder);
        } else {
            view = convertView;
            viewHolder = (ViewHolder) view.getTag();
        }

        viewHolder.image.setImageResource(fruit.getFruitImage());
        viewHolder.text.setText(fruit.getFruitName());

        return view;
    }
}
```

ListDemoAdapter.java

- ***Declare*** *ListView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with ListView in the Activity*

List View

```java
// To bind ListView and RecyclerView to the corresponding layout
private void initializeList(View view) {
    // To bind ListView adapter to ListView
    if (this.layout == R.layout.fragment_layout_demo_list) {
        ListDemoAdapter adapter = new ListDemoAdapter(getActivity(), R.layout.list_example, getFruits());
        listView = view.findViewById(R.id.demo_list_view);
        listView.setAdapter(adapter);

        // To set onItemClickListener - method 1
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
                Fruit fruit = (Fruit)adapterView.getItemAtPosition(i);
                Toast.makeText(getContext(), fruit.getFruitName(), Toast.LENGTH_SHORT).show();
            }
        });
    }
```

LayoutFragmentDemo.java

- ***Declare*** *ListView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with ListView in the Activity*

List View – Listeners when Clicking Items


```java
if (this.layout == R.layout.fragment_layout_demo_list) {
    ListDemoAdapter adapter = new ListDemoAdapter(getActivity(), R.layout.list_example, getFruits());
    listView = view.findViewById(R.id.demo_list_view);
    listView.setAdapter(adapter);

    // To set onItemClickListener - method 1
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
            Fruit fruit = (Fruit) adapterView.getItemAtPosition(i);
            Toast.makeText(getContext(), fruit.getFruitName(), Toast.LENGTH_SHORT).show();
        }
    });
}
```

LayoutFragmentDemo.java

```java
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    Fruit fruit = getItem(position);
    View view;
    ViewHolder viewHolder;
    if (convertView == null) {
        view = LayoutInflater.from(getContext()).inflate(resourceId, root: null);
        viewHolder = new ViewHolder(view);
        view.setTag(viewHolder);
    } else {
        view = convertView;
        viewHolder = (ViewHolder) view.getTag();
    }

    Set onClick listener - method 2
    view.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Toast.makeText(getContext(), ((ViewHolder) view.getTag()).text.getText(), Toast.LENGTH_SHORT).show();
        }
    });

    viewHolder.image.setImageResource(fruit.getFruitImage());
    viewHolder.text.setText(fruit.getFruitName());

    return view;
}
```

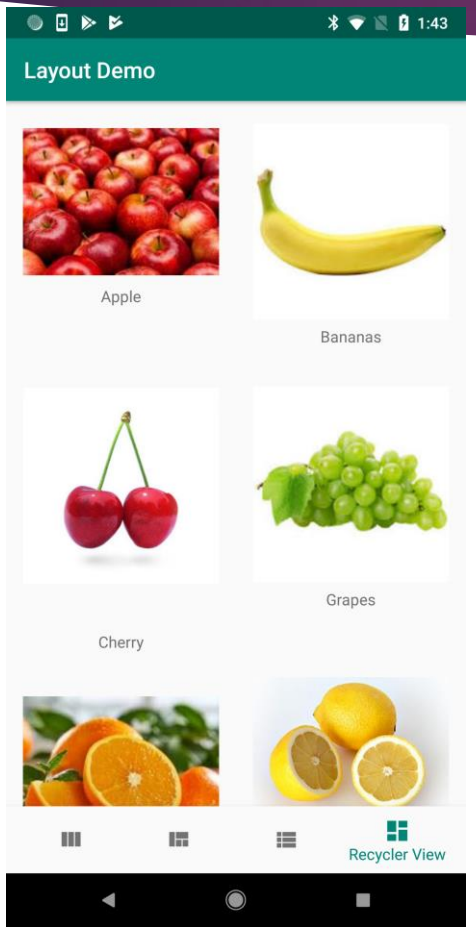
ListDemoAdapter.java

User Interface

- *Recycler View*

(A more advanced and flexible version of ListView)

Recycler View



To create a ***Recycler View***:

- ***Declare*** *RecyclerView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with RecyclerView in the Activity*

https://developer.android.com/guide/topics/ui/layout/recyclerview

Recycler View

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".LayoutDemoFragment">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/demo_recycler"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

fragment_layout_demo_recycler.xml

- ***Declare*** *RecyclerView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with RecyclerView in the Activity*

Recycler View

```java
package com.example.layoutdemo;


// Fruit class contains fruitImage and fruitName
// With automatically generated Setter and Getter
public class Fruit {
    // fruitImage to store the resource id if fruit image
    private int fruitImage;
    // fruitName to store the string of fruit name
    private String fruitName;

    public Fruit(int fruitImage, String fruitName) {
        this.fruitImage = fruitImage;
        this.fruitName = fruitName;
    }

    public int getFruitImage() { return fruitImage; }

    public void setFruitImage(int fruitImage) { this.fruitImage = fruitImage; }

    public String getFruitName() { return fruitName; }

    public void setFruitName(String fruitName) { this.fruitName = fruitName; }
}
```

Fruit.java

- ***Declare*** *RecyclerView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with RecyclerView in the Activity*

Recycler View

```java
//RecyclerView Adapter for Fruits
public class RecyclerDemoAdapter extends RecyclerView.Adapter<RecyclerDemoAdapter.ViewHolder> {
    static class ViewHolder extends RecyclerView.ViewHolder {
        // to bind with ImageView from R.layout.recycler_example.xml
        // to show the fruit image
        @BindView(R.id.list_example_image)
        ImageView image;
        // to bind with TextView from R.layout.recycler_example.xml
        // to show the fruit name
        @BindView(R.id.list_example_text)
        TextView text;

        ViewHolder(View view) {
            super(view);

            //...
            view.setOnClickListener(new View.OnClickListener() {...});

            ButterKnife.bind( target: this, view);
        }

    }

    // an array of fruits need to display at recyclerView
    private List<Fruit> fruits;
    // the resource id of item layout
    private int resourceId;

    // to initialize adapter with fruits array, and the resource id of layout
    public RecyclerDemoAdapter(List<Fruit> fruits, int resourceId) {
        this.fruits = fruits;
        this.resourceId = resourceId;
    }

    @NonNull
    @Override
    // Called when RecyclerView needs a new {@link ViewHolder} of the given type to represent an item
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View v = LayoutInflater.from(parent.getContext().inflate(resourceId, parent, attachToRoot: false);
        ViewHolder vh = new ViewHolder(v);
        return vh;
    }

    @Override
    // to bind the resources to viewHolder, including fruit image resource id and fruit name
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        holder.image.setImageResource(fruits.get(position).getFruitImage());
        holder.text.setText(fruits.get(position).getFruitName());
    }

    @Override
    // to get the size of the fruits array
    public int getItemCount() { return fruits.size(); }

}
```

RecyclerDemoAdapter.java

- ***Declare*** *RecyclerView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with RecyclerView in the Activity*

Recycler View


```
RecyclerDemoAdapter adapter = new RecyclerDemoAdapter(getFruits(), R.layout.recycler_example);
recyclerView = view.findViewById(R.id.demo_recycler);
// To lay out children in a staggered grid formation
recyclerView.setLayoutManager(new StaggeredGridLayoutManager( spanCount: 2, StaggeredGridLayoutManager.VERTICAL));
recyclerView.setAdapter(adapter);
```

LayoutDemoFragment.java

- ***Declare*** *RecyclerView in layout (XML)*
- ***Create*** *Object*
- ***Create*** *Adapter and ViewHolder*
- ***Bind*** *Adapter with RecyclerView in the Activity*

Recycler View – Listener when Clicking Items

```java
static class ViewHolder extends RecyclerView.ViewHolder {
    // to bind with ImageView from R.layout.recycler_example.xml
    // to show the fruit image
    @BindView(R.id.list_example_image)
    ImageView image;
    // to bind with TextView from R.layout.recycler_example.xml
    // to show the fruit name
    @BindView(R.id.list_example_text)
    TextView text;

    ViewHolder(View view) {
        super(view);

        // To show how to add click listener to a item in recyclerView
        // Set onClickListener for the fruit array;
        // When clicking a item from the list, a new Toast shows the name of the clicked fruit;
        view.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(view.getContext(), text.getText(), Toast.LENGTH_SHORT).show();
            }
        });

        ButterKnife.bind( target: this, view);
    }

}
```

RecyclerDemoAdapter.java

User Interface

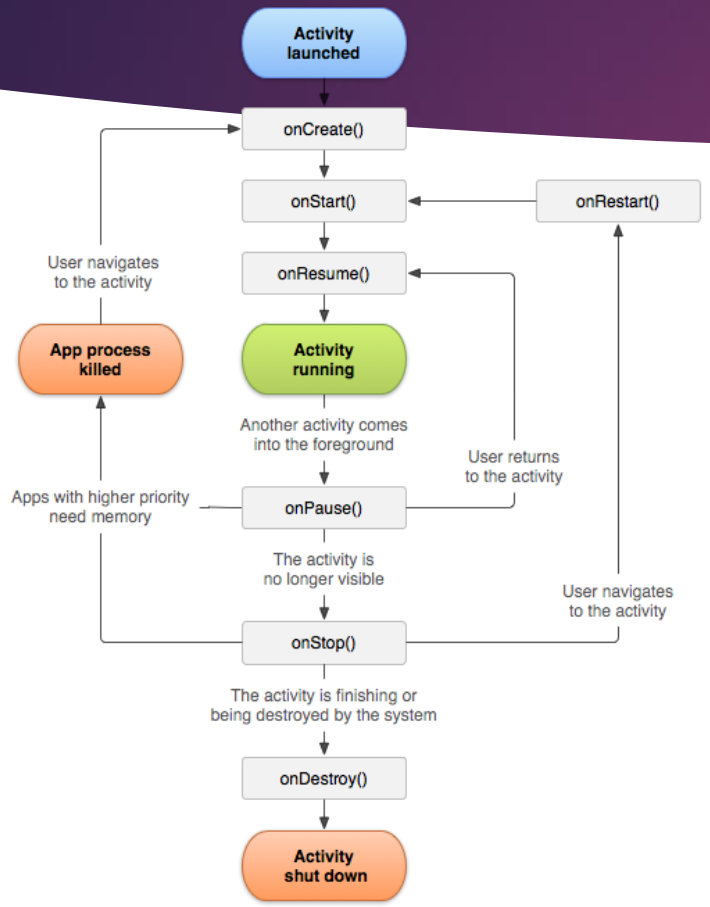
- *Activity*

Activity

*An **activity** provides the window in which the app draws its **UI**.*

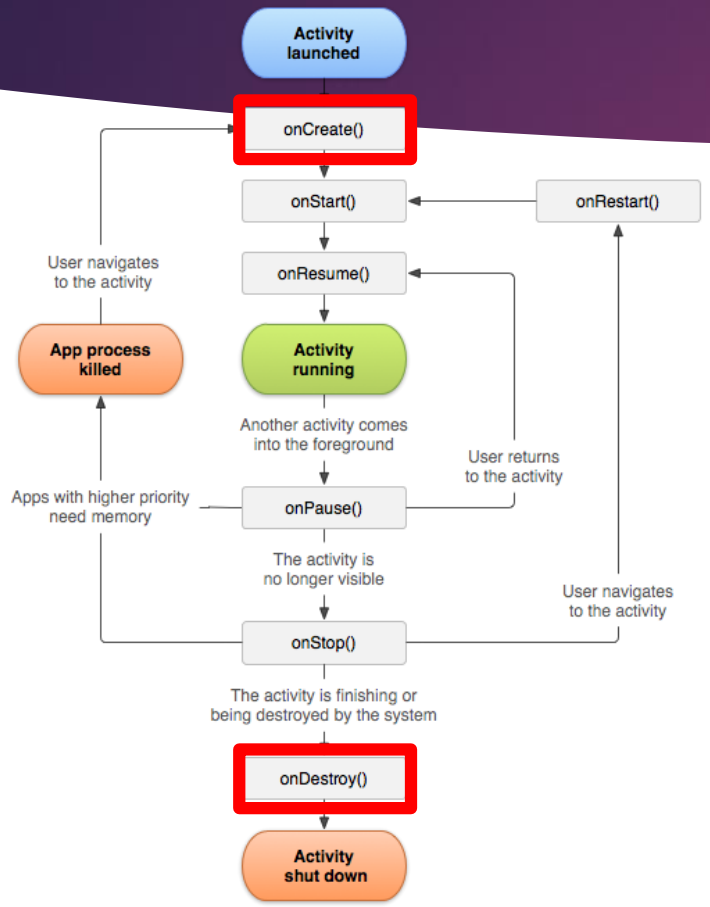
https://developer.android.com/guide/components/activities/intro-activities

Activity



Unlike programming paradigms in which apps are launched with a main() method, the Android system ***initiates code in an Activity instance*** by ***invoking specific callback methods*** that correspond to ***specific stages of its lifecycle***.

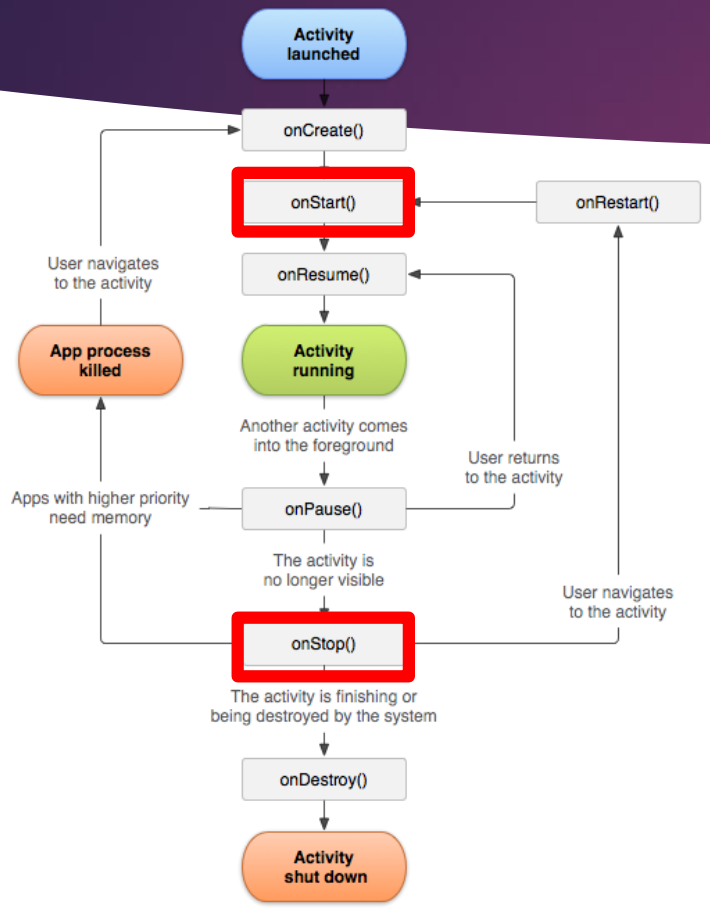
Activity – onCreate/onDestroy



The ***entire lifetime*** of an activity happens between the first call to ***onCreate(Bundle)*** through to a single final call to ***onDestroy()***.

An activity will do all ***setup of "global" state in onCreate()***, and ***release all remaining resources in onDestroy()***.

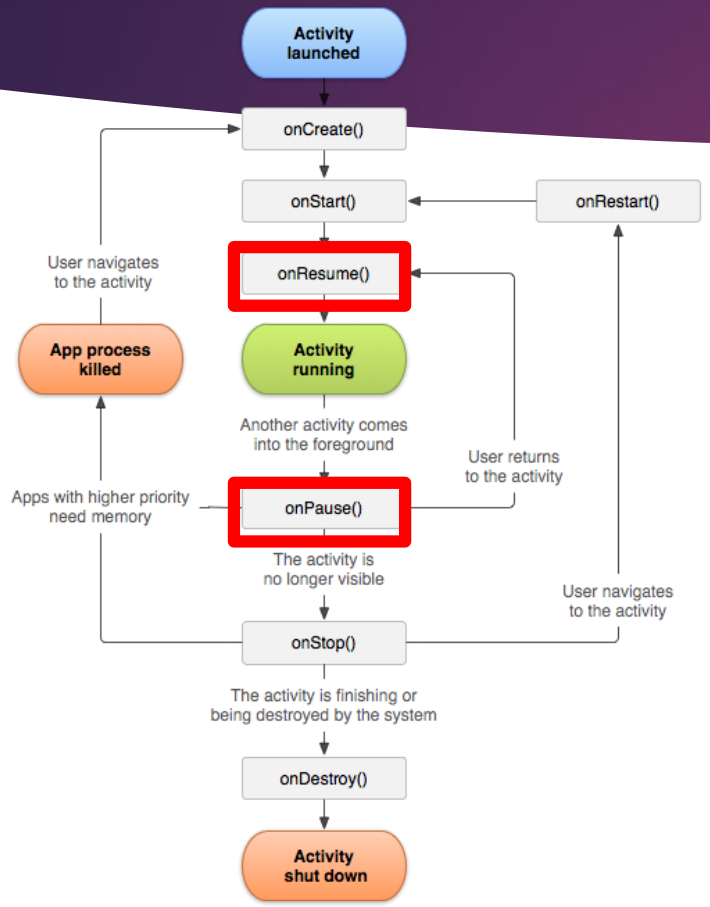
Activity – onStart/onStop



The ***visible lifetime*** of an activity happens between a call to ***onStart()*** until a corresponding call to ***onStop()***.

During this time the user can see the activity ***on-screen***, though it ***may not be in the foreground*** and interacting with the user.

Activity – onResume/onPause



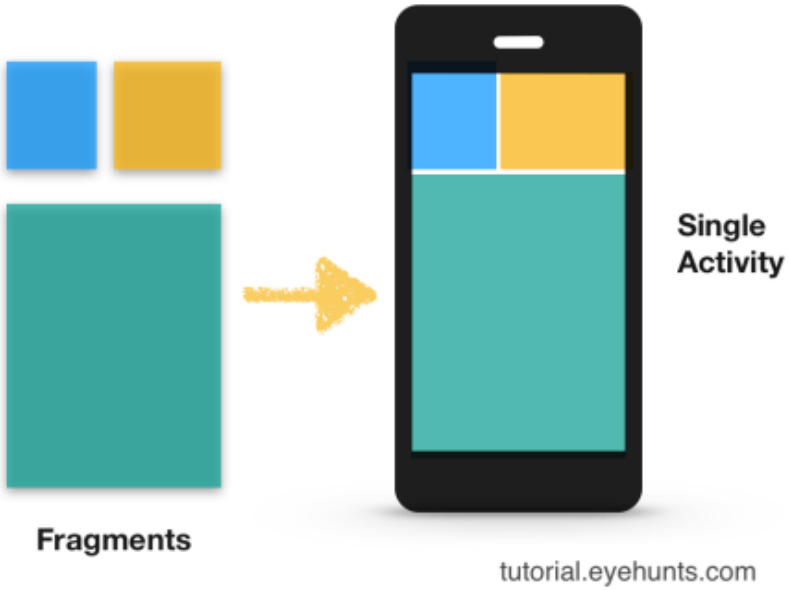
The ***foreground lifetime*** of an activity happens between a call to ***onResume()*** until a corresponding call to ***onPause().***

During this time the activity is in ***visible, active and interacting*** with the user.

User Interface

- *Fragment*

Fragment

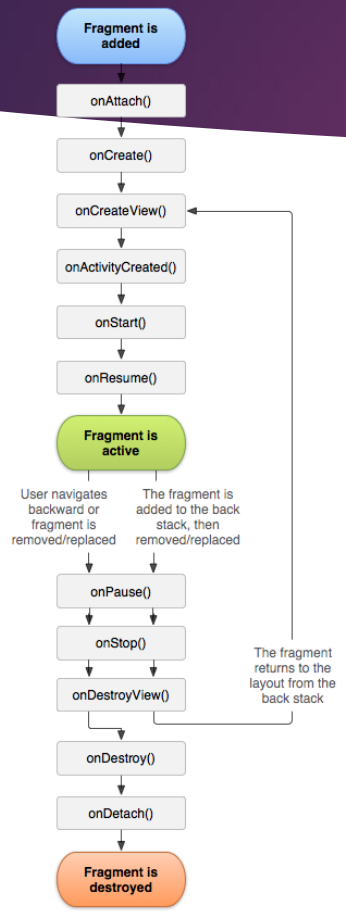


A Fragment represents ***a behaviour or a portion of user interface*** in a *FragmentActivity*.

You can ***combine multiple fragments in a single activity*** to build a multi-pane UI and ***reuse a fragment in multiple activities***.

https://developer.android.com/guide/components/fragments

Fragment



You can think of a fragment as ***a modular section*** of an activity, which *has its own lifecycle, receives its own input events,* and which you can *add or remove while the activity is running* (sort of like a "sub activity" that you can reuse in different activities).

Fragment – adding a Fragment to an Activity


```
<fragment
    android:id="@+id/layout_fragment"
    android:name="com.example.layoutdemo.LayoutDemoFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_above="@+id/nav_view"
    android:orientation="vertical"
    />
```

Declare the fragment inside the activity's layout file(XML)

- In this case, you can specify layout properties for the fragment as if it were a view.

Fragment – adding a Fragment to an Activity


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

    // Setting for Navigation Bar
    BottomNavigationView navView = findViewById(R.id.nav_view);
    navView.setOnNavigationItemSelectedListener(mOnNavigationItemSelectedListener);

    // Setting for Fragments
    Fragment linear_layout = LayoutDemoFragment.newInstance(LayoutDemoFragment.LINEAR_DEMO);
    getSupportFragmentManager() FragmentManager
            .beginTransaction() FragmentTransaction
            .replace(R.id.layout_fragment, linear_layout) FragmentTransaction
            .addToBackStack(null) FragmentTransaction
            .commit();

    ButterKnife.bind( target: this);
}
```

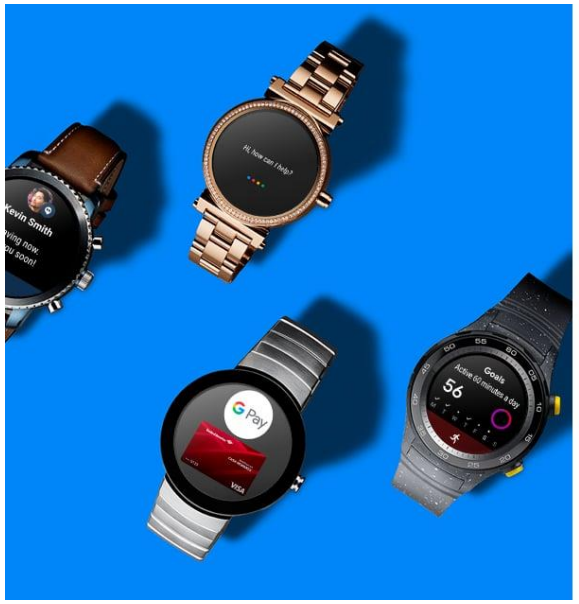
Programmatically add the fragment to an existing ViewGroup

- At any time while your activity is running, you can add fragments to your activity layout.

Android

Wear OS

Android Wear OS



Wear OS by Google

Wear OS, also known as Wear OS by ***Google*** and previously called Android Wear, is a version of Google's Android operating system designed for ***smartwatches and other wearables***.

https://developer.android.com/training/wearables

Android Wear OS – Applications

GOOGLE KEEP

Keep is easily ***note-taking apps***, which is designed for quick reminders and shopping lists

Android Wear OS – Applications

WATER DRINK REMINDER

The app knows ***how much*** you need to ***drink at regular intervals throughout the day***.

Android Wear OS – Applications

PARKING

ParKing lets you record ***where you parked your car***.

Android Wear OS – Applications

SEVEN MINUTE WORKOUT

Seven Minute Workout is a quick, neat way to ***get a guided workout*** before you go to work.

Android Wear OS – Source Code


```java
public class MainActivity extends WearableActivity {

    private TextView mTextView;

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

        mTextView = (TextView) findViewById(R.id.text);

        // Enables Always-on
        setAmbientEnabled();
    }
}
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.wear.widget.BoxInsetLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/dark_grey"
    android:padding="0dp"
    tools:context=".MainActivity"
    tools:deviceIds="wear">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="5dp"
        app:boxedEdges="all">

        <TextView
            android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello Square World!" />

    </FrameLayout>
</androidx.wear.widget.BoxInsetLayout>
```

Android Wear OS – Demonstration

Android Wear OS – Debugging mode

Enable adb debugging on the watch:

1. Open the ***Settings*** menu on the watch.
2. Scroll to the bottom of the menu. If no ***Developer options*** item is provided, tap ***System*** and then ***About***.
3. Tap the ***build number*** 7 times.
4. From the ***Settings*** menu, tap ***Developer options***.
5. Enable ***ADB debugging***.

https://developer.android.com/training/wearables/apps/creating

THE UNIVERSITY OF MELBOURNE

Thank you