


ACCESS RESOURCES

Resource Type	In Java code	In XML
Image 	R.drawable.photo	@drawable/photo
String "Hello"	R.string.hello	@string/hello
Layout XML file	R.layout.activity_main	@layout/activity_main
ID	R.id.price_text_view	@id/price_text_view
Color #FF0000	R.color.red	@color/red

Create Object

- New
- Factory method
- Padding inside the view “android: padding”
- Margin outside the view “Android: layout_margin”
- Layout_gravity **Outside** gravity
- Gravity **Inside** gravity

Layout -> outside view relationship

This image is used when you pass parameter in middle of string or concatenate words

Parameters

“Received at 2 o’clock”

```
<string name="received_at_oclock">
  Received at
    <xliff:g id="time" example="5">%1$s</xliff:g>
  o'clock
</string>
```

Style vs Theme

- Style on single view
- Theme on activity or app

GNU stands for “GNU's Not Unix”.

Unix stands for Uniplexed Information and Computer Systems.

Build Tool

➔ are programs that automate the creation of executable applications from source code(eg. .apk for android app). Building incorporates compiling,linking and packaging the code into a usable or executable form.

build automation

➔ In small projects, developers will often manually invoke the build process. This is not practical for larger projects, where it is very hard to keep track of what needs to be built, in what sequence and what dependencies there are in the building process. Using an automation tool allows the build process to be more consistent.

adb

➔ Android Debug Bridge

Android App

➔ Collection of Connected Components works each other with android framework

Four Types of Component Registered on android Manifest

- 1- Activity
- 2- Services
- 3- Content Provider
- 4- Broad Cast Receiver

Type of View: UI Components

There are two major categories of views. The first type are UI components that are often interactive. Here are a few examples:

Class Name	Description
TextView	Creates text on the screen; generally non interactive text.
EditText	Creates a text input on the screen
ImageView	Creates an image on the screen
Button	Creates a button on the screen
Chronometer	Create a simple timer on screen

The [android.widget](#) package contains a list of *most* of the UI view classes available to you.

Type of View : Container View

The second are views called "Layout" or "Container" views. They extend from a class called [ViewGroup](#). They are primarily responsible for containing a group of views and determining where they are on screen. What do I mean by "containing a group of views?". I mean that a view will be nested inside the tag of another view, like below:

Class Name	Description
LinearLayout	Displays views in a single column or row.
RelativeLayout	Displays views positioned relative to each other and this view.
FrameLayout	A ViewGroup meant to contain a single child view.
ScrollView	A FrameLayout that is designed to let the user scroll through the content in the view.
ConstraintLayout	This is a newer viewgroup; it positions views in a flexible way. We’ll be exploring constraint layout later in the lesson.

The R Class

When your application is compiled the [R](#) class is generated. It creates constants that allow you to dynamically identify the various contents of the `res` folder, including layouts. To learn more, check out the documentation about [resources](#).

Short cuts

Android Comment -> Ctrl + /

Arrange XML -> Ctrl + Alt + L

Prediction -> Ctrl + Space

Ctrl + O -> show the available methods to override

ALT + Insert -> automatically “Generate a method (Getters, Setters, Constructors, toString, etc..)”.

On Windows

@+id/tv_toy_names

@ -> look at resources

+ -> create if doesn’t exist

id -> create id not string, style nor image

ConstraintLayout

➔ Special kind of layout uses complex constrain on child views to allow for dynamic layoutsrespect different window sizes layouts

Handling different screen -> Using **Infer Constraints “inference”**

```
<string name="today">Today</string>

<!-- For labelling tomorrow's forecast [CHAR LIMIT=15] -->
<string name="tomorrow">Tomorrow</string>

<!-- Date format [CHAR LIMIT=NONE] -->
<string name="format_full_friendly_date">
    <xliff:g id="month">%1$s</xliff:g>, <xliff:g id="day">%2$s</xliff:g>
</string>
```

The id of the String with the value "Today" is **today** and the id of the String with the value **<xliff:g id="month">%1\$s</xliff:g>, <xliff:g id="day">%2\$s</xliff:g>** is **format_full_friendly_date**

If you wanted to reference the **Today** string, you would reference it in Java by doing something like this:

```
String myString = getString(R.string.today);
```

In XML, you can access a String by using the @string accessor method. For the same String defined above, you could access it like this:

```
<TextView text="@string/today" />
```

Menu

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main,menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int menuItemThatWasSelected = item.getItemId();
    if(menuItemThatWasSelected == R.id.action_search)
    {
        Context context = MainActivity.this;
        String message = "Search clicked";
        Toast.makeText(context,message,Toast.LENGTH_LONG).show();
    }
    return super.onOptionsItemSelected(item);
}
```

URL

```
public static URL buildUrl(String githubSearchQuery) { //Metod in class NetworkUtils.java
    // TODO (1) Fill in this method to build the proper Github query URL
    Uri builtUri = Uri.parse(GITHUB_BASE_URL).buildUpon()
        .appendQueryParameter(PARAM_QUERY, githubSearchQuery)
        .appendQueryParameter(PARAM_SORT, sortBy)
        .build();
    URL url = null;
    try {
        url = new URL(builtUri.toString());
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
    return url;
}

//MainActivity

private EditText mSearchBoxEditText;

private TextView mUrlDisplayTextView;

private TextView mSearchResultsTextView;

protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main);

mSearchBoxEditText = (EditText) findViewById(R.id.et_search_box);

mUrlDisplayTextView = (TextView) findViewById(R.id.tv_url_display);

mSearchResultsTextView = (TextView) findViewById(R.id.tv_github_search_results_json);

}

private void makeGithubSearchQuery() {

String githubQuery = mSearchBoxEditText.getText().toString();

URL githubSearchUrl = NetworkUtils.buildUrl(githubQuery);

mUrlDisplayTextView.setText(githubSearchUrl.toString());

}
```

Implicit intent

- ➔ Like open link on any browser you don't know where to go

Explicit intent

- ➔ From activity to another you know where exactly to go usually on same app

WHAT'S INSIDE AN INTENT?

Implicit Intent

- Action
- Data URI
- Category
- Components
- Extras

WHAT'S INSIDE AN INTENT?

Explicit Intent

- Data URI
- Context
- Component (usually Class/Activity)

COMPARE & CONTRAST

Implicit Intent

```
// Create the text message with a string
Intent intent = new Intent(Intent.ACTION_SENDTO);
sendIntent.setData(Uri.parse("mailto:"));
sendIntent.putExtra(Intent.EXTRA_SUBJECT, "Just Java order for " + name);
sendIntent.putExtra(Intent.EXTRA_TEXT, priceMessage);

// Verify that the intent will resolve properly
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

IMPLICIT INTENT

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

EXPLICIT INTENT

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

Explicit Intent

```
// Executed in an Activity, so 'this' is the Context
Intent intent = new Intent(this, NumbersActivity.class);
startActivity(intent);
```

Concrete Class	Abstract Class	Interface
<pre>public class TextView { String mText; int mTextColor; void setText (String text) { mText = text; } void setTextColor (int color) { mTextColor = color; } . . . }</pre>	<pre>public abstract class ViewGroup int mChildrenCount; void addView (View child) addView (child, -1); void removeView (View view) removeViewInternal (view); void onLayout (); . . . }</pre>	<pre>public interface OnClickListener void onClick (View v);</pre>
Fully Implemented	Partially Implemented	Not implemented at all
<ul style="list-style-type: none">- Contains state- Methods are fully implemented	<ul style="list-style-type: none">- Contains state- Some methods are fully implemented- Some methods are abstract (not implemented)	<ul style="list-style-type: none">- No state- All methods are abstract (not implemented)

Java Final Keyword

⇒ Stop Value Change

⇒ Stop Method Overriding

⇒ Stop Inheritance

javatpoint.com

Static

- 1) Java static variable
 - The static variable gets memory only once in class area at the time of class loading.
 - Java static property is shared to all objects.
- 2) Java static method If you apply static keyword with any method, it is known as static method.
 - A static method belongs to the class rather than object of a class.
 - A static method can be invoked without the need for creating an instance of a class.
 - static method can access static data member and can change the value of it.
- 3) Java static block
 - Is used to initialize the static data member.
 - It is executed before main method at the time of classloading.

Restrictions for static method There are two main restrictions for the static method. They are:

- The static method **cannot** use non-static data member or call non-static method directly.
- **this** and **super** cannot be used in static context.

An empty interface is known as tag or marker interface.

Interface

- can't instantiate an interface in java
- **full abstraction**
- class **implements** interface, but an interface **extends** another interface.
- Interface can't be declared as **private**, **protected** or **transient**
- Methods are by default **abstract** and **public**
- Variables are by default **public**, **static** and **final**
- Interface **variables must be initialized at the time of declaration** otherwise compiler will throw an error.
- Class can implement any number of interfaces
- If there are **two or more same methods** in two interfaces and a class implements both interfaces, implementation of the method once is enough
- A class cannot implement two interfaces that have methods with same name but different return type.
- Variable names conflicts can be resolved by interface name.

Advantages of interface in java

- Without bothering about the implementation part, we can achieve the security of implementation
- In java, **multiple inheritance** is not allowed, however you can use interface to make use of it as you can implement more than one interface.

Object Oriented Approach OOP

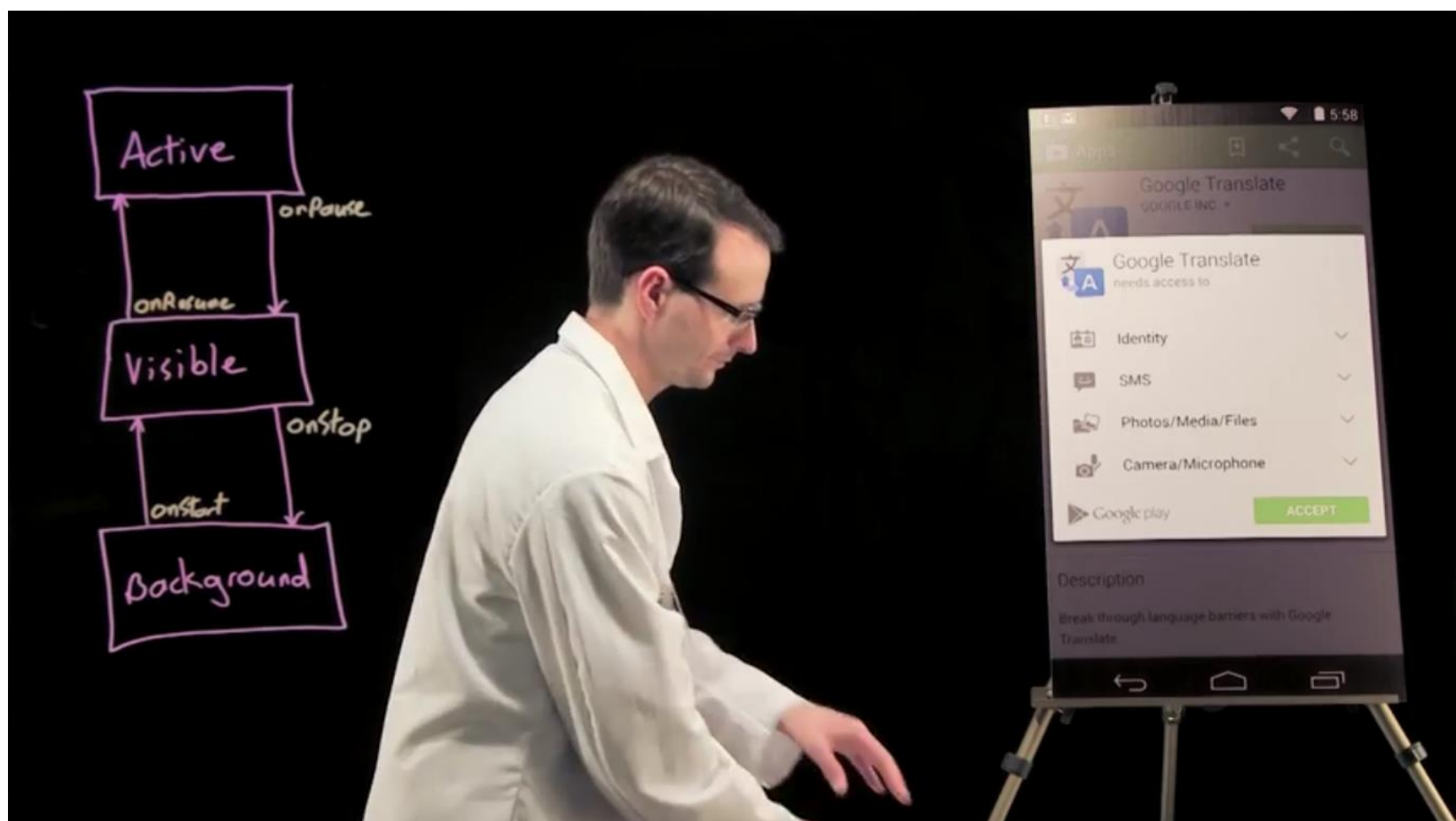
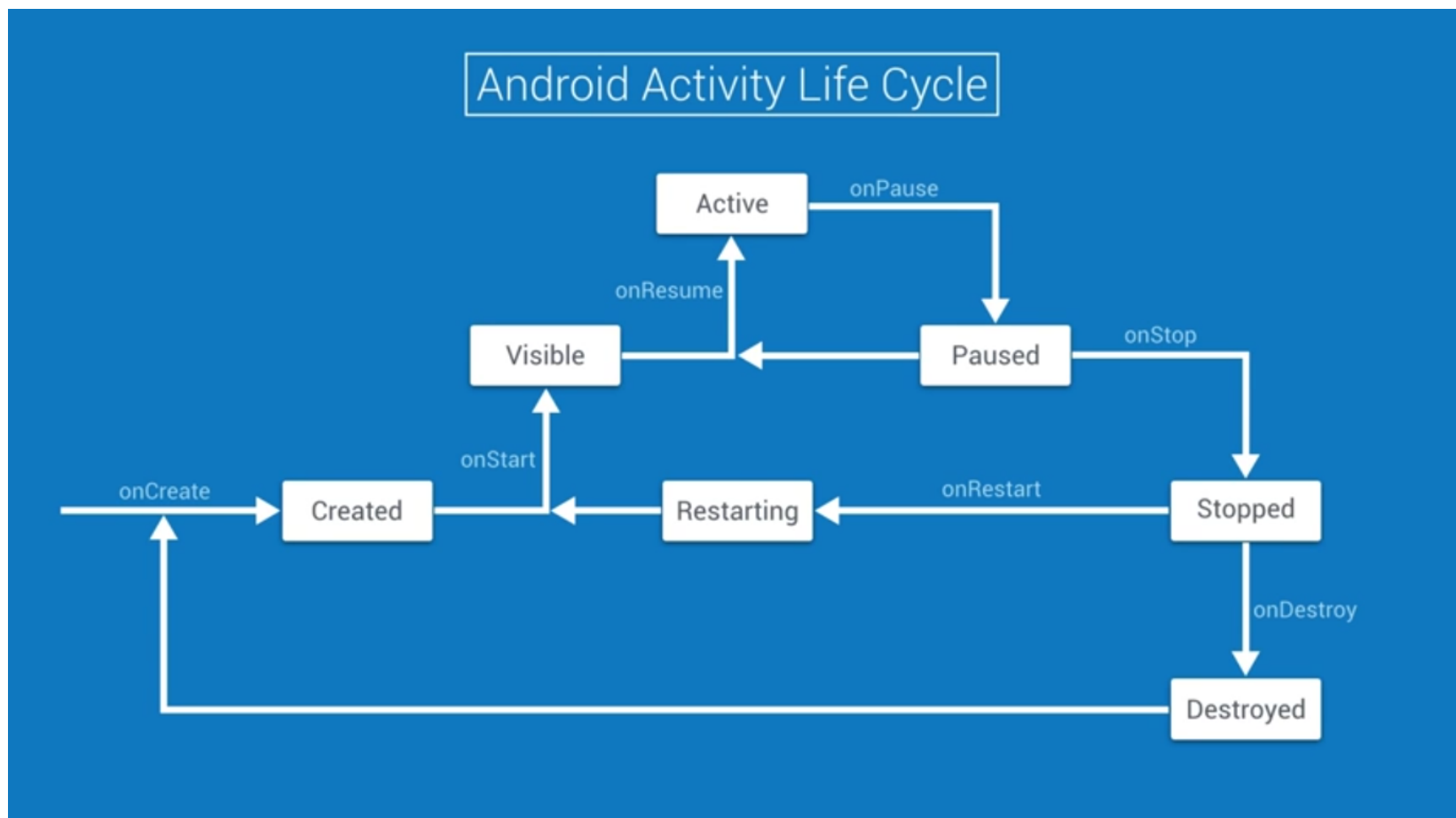
- **Abstraction**
 - **show only “relevant” data and “hide” unnecessary details of an object from the user.**
- **Encapsulation**
 - **binding (ربط) object state(fields) and behavior (methods) together. If you are creating class, you are doing encapsulation.**
 - The idea of encapsulation is to keep classes separated and prevent them from having tightly coupled with each other.
 - The whole idea behind encapsulation is to **hide the implementation details from users**. If a **data member is private it means it can only be accessed within the same class**. No outside class can access private data member (variable) of other class. But you can use public setter and getters to access
 - Binding the data with the code that manipulates it.
 - It keeps the data and the code safe from external interference
 - Encapsulated code should have following characteristics:
 - Everyone knows how to access it.
 - Can be easily used regardless of implementation details.
 - There shouldn't any side effects of the code, to the rest of the application.
- **Inheritance**
 - **The process by which one class acquires (اكتساب) the properties and functionalities of another class is called inheritance.**
 - Inheritance is the mechanism by which an object **acquires** the some/all properties of another object.
 - It supports the concept of hierarchical classification.
- **Polymorphism**
 - **allows us to perform a single action in different ways.** For example, let's say we have a class `Animal` that has a method `animalSound()`, here we cannot give implementation to this method as we do not know which `Animal` class would extend `Animal` class. So, we make this method abstract like this:
 - Polymorphism could be static and dynamic both. Method **Overloading is static polymorphism while, Method overriding is dynamic polymorphism.**
 - Polymorphism means to process objects differently based on their data type.
 - In other words, it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)
 - This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.
- **Overloading** in simple words means more than one method having the same method name that behaves differently based on the arguments passed while calling the method. This called static because, which method to be invoked is decided at the time of compilation

- **Overriding** means a derived class is implementing a method of its super class. The call to overridden method is resolved at runtime, thus called runtime polymorphism

Generalization is the process of extracting shared characteristics from two or more classes, and combining them into a generalized superclass.

specialization means creating new subclasses from an existing class

Android Activity Life Cycle



Event Listener

HOW TO SETUP AN EVENT LISTENER

1. Define an **event listener** (and the custom behavior for when the event happens)

In NumbersActivity.java:

```
public class NumbersClickListener implements OnClickListener {  
    @Override  
    public void onClick(View view) { Toast.makeText(view.getContext(),  
        "Open the list of numbers", Toast.LENGTH_SHORT).show();}  
}
```

2. Create a **new object instance** of the event listener (using the constructor)

In MainActivity.java:

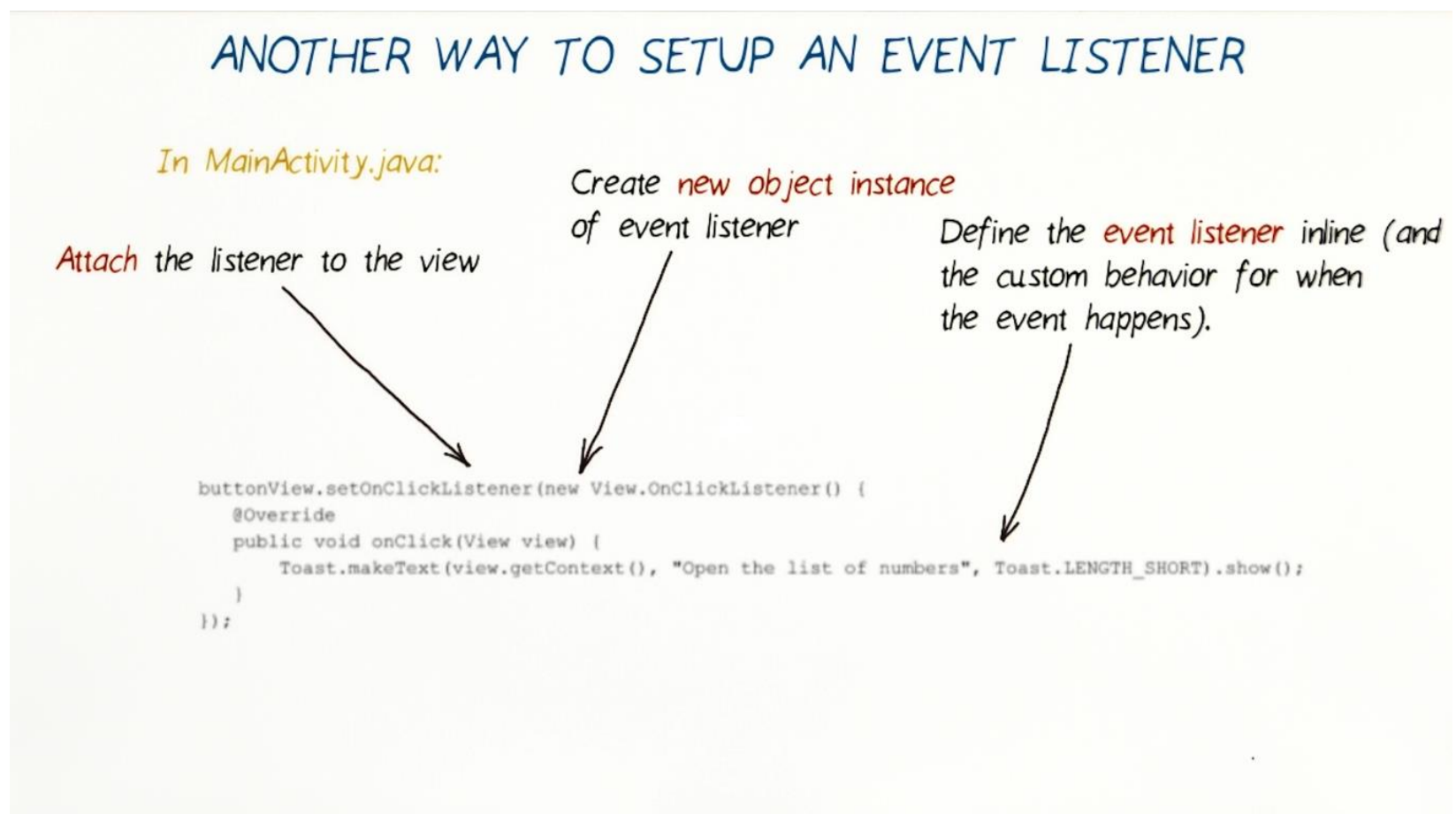
```
NumbersClickListener clickListener = new NumbersClickListener();
```

3. **Attach** the listener to the view

In MainActivity.java:

```
buttonView.setOnClickListener(clickListener);
```

Event Listener in one line



OnClickListener vs onClick

You might be wondering why we're going through all the trouble of creating an anonymous subclass of `OnClickListener` and attaching it to a view, when we already know how to use the `onClick` XML attribute from from back in [Android Basics: User Input](#). Why write something terrifying like:

```
// In onCreate() in the Activity  
Button button = (Button) findViewById(R.id.ze_button);  
button.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        doSomeStuff();  
    }  
});
```

When we could do something much cleaner like:

```
android:onClick="myListener" // This is in the XML layout  
  
public void myListener(View view){ // This is back in the Activity file  
    doSomeStuff();  
}
```

There are a couple reasons why you might want to programmatically set an `OnClickListener`. The first is if you ever want to change the behavior of your button while your app is running. You can point your button at another method entirely, or just disable the button by setting an `OnClickListener` that doesn't do anything.

When you define a listener using the `onClick` attribute, the view looks for a method with that name only in its host activity. Programmatically setting an `OnClickListener` allows you to control a button's behavior from somewhere other than its host activity. This will become very relevant when we learn about `Fragments`, which are basically mini activities, allowing you to build reusable collections of views with their own lifecycle, which can then be assembled into activities. `Fragments` always need to use `OnClickListener`s to control their buttons, since they're not `Activities`, and won't be searched for listeners defined in `onClick`.

For more commentary on the decision to use an `OnClickListener` vs `onClick`, check out [this question](#) on Stack Overflow.



ArrayList

HOW TO CREATE AND ACCESS ELEMENTS IN AN ARRAYLIST

Create an ArrayList

```
ArrayList<String> musicLibrary = new ArrayList<String>();
```

Add elements in an ArrayList

```
musicLibrary.add("Yellow Submarine");
musicLibrary.add("Thriller");

// Adds an element at a specific index
musicLibrary.add(0, "Blue Suede Shoes");
```

Access elements in an ArrayList

```
musicLibrary.get(0);
musicLibrary.get(1);
musicLibrary.get(2);
```

Remove elements from an ArrayList

```
// Removes the element at the specific index
musicLibrary.remove(2);
```

Get the ArrayList length or size

```
musicLibrary.size();
```

GCE Garbage Collection Event

ArrayAdapter

```
ArrayAdapter<String> itemsAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, words);
```

```
ListView listView = (ListView) findViewById(R.id.list);
```

```
listView.setAdapter(itemsAdapter);
```

Custom Adapter

Word contain structure words default language and other or images etc..

WordAdapter extends ArrayAdapter<Word> and override getView method

```
ArrayList<Word> words= new ArrayList<Word>();
words.add(new Word("one","lutti"));
```

```
WordAdapter adapter = new WordAdapter(this,words); // consider as ArrayAdapter but customized for usage
ListView listView = (ListView) findViewById(R.id.list);
listView.setAdapter(adapter);
```

```
.....
public class WordAdapter extends ArrayAdapter<Word> {
```

```
    public WordAdapter(@NonNull Context context, ArrayList resource) {
        super(context, 0,resource);
    }
```

```
    @Override
```

```
    public View getView(int position, View convertView, ViewGroup parent) {
```

```
        // Check if the existing view is being reused, otherwise inflate the view
```

```
        View listItemView = convertView;
```

```
        if(listItemView == null) {                                //list_item.xml this have two text view content that you want
```

```
            listItemView = LayoutInflater.from(getContext()).inflate(R.layout.list_item, parent, false);
```

```
        }
```

```
        Word local_word = getItem(position);
```

```
        TextView miwokTextView = (TextView) listItemView.findViewById(R.id.miwok_text_view);
```

```
        miwokTextView.setText(local_word.getMiwokTranslation());
```

```
        TextView defaultTextView = (TextView) listItemView.findViewById(R.id.default_text_view);
```

```
        defaultTextView.setText(local_word.getDefaultTranslation());
```

```
        return listItemView;
```

```
    }
```

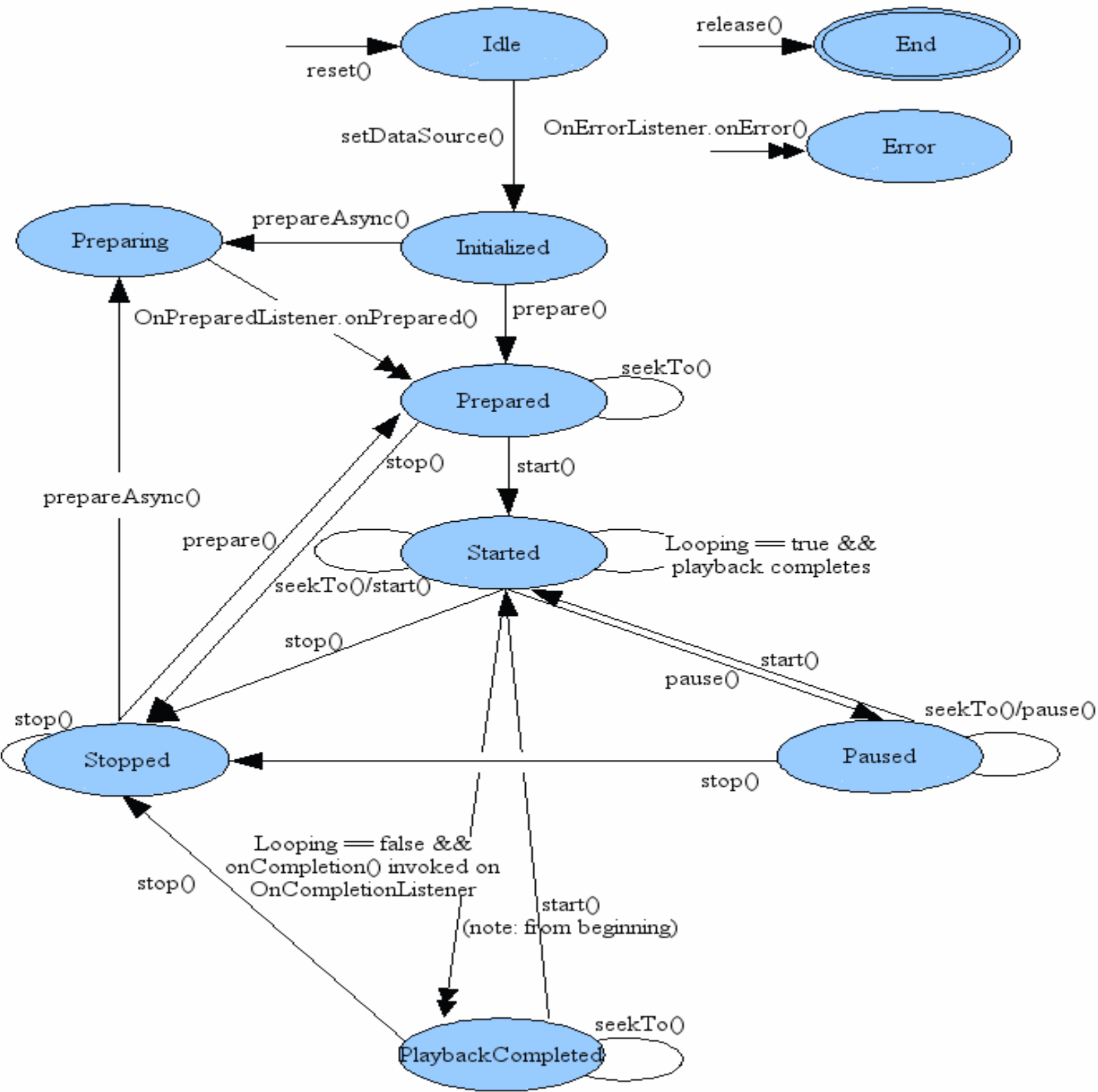
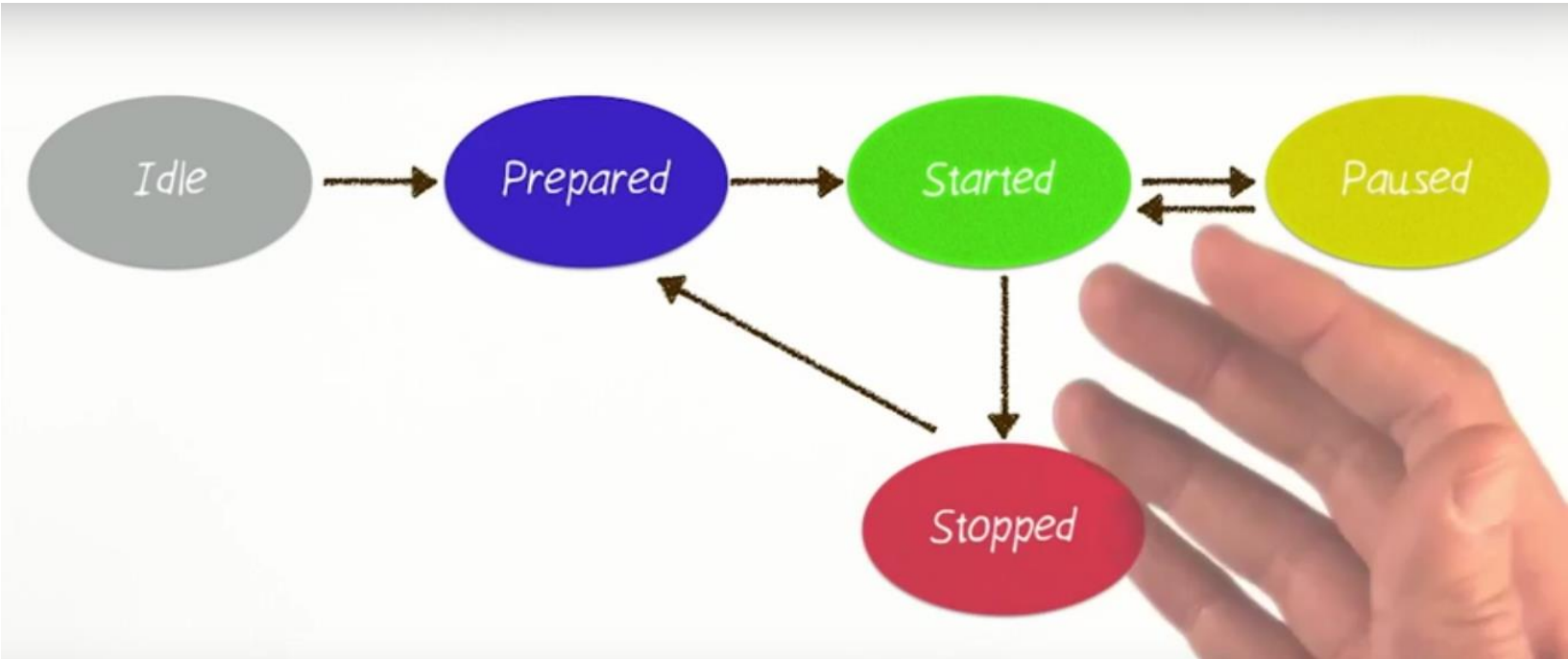
```
}
```

```
.....
UNIX -> Uniplexed Information and Computer Systems.
```

It was called Multics for Multiplexed Information and Computing System. *Unix* was named after multics and was intended as sort of an insult since Multics had become too complex

Note drawable name must be lowercase,0-9 or underscore

Media Player States



Note

When you use image in app you must past It inside android studio IDE not in explorer file be careful

AdapterView

- ListView
- GiridView
- Spinner
- Gallery

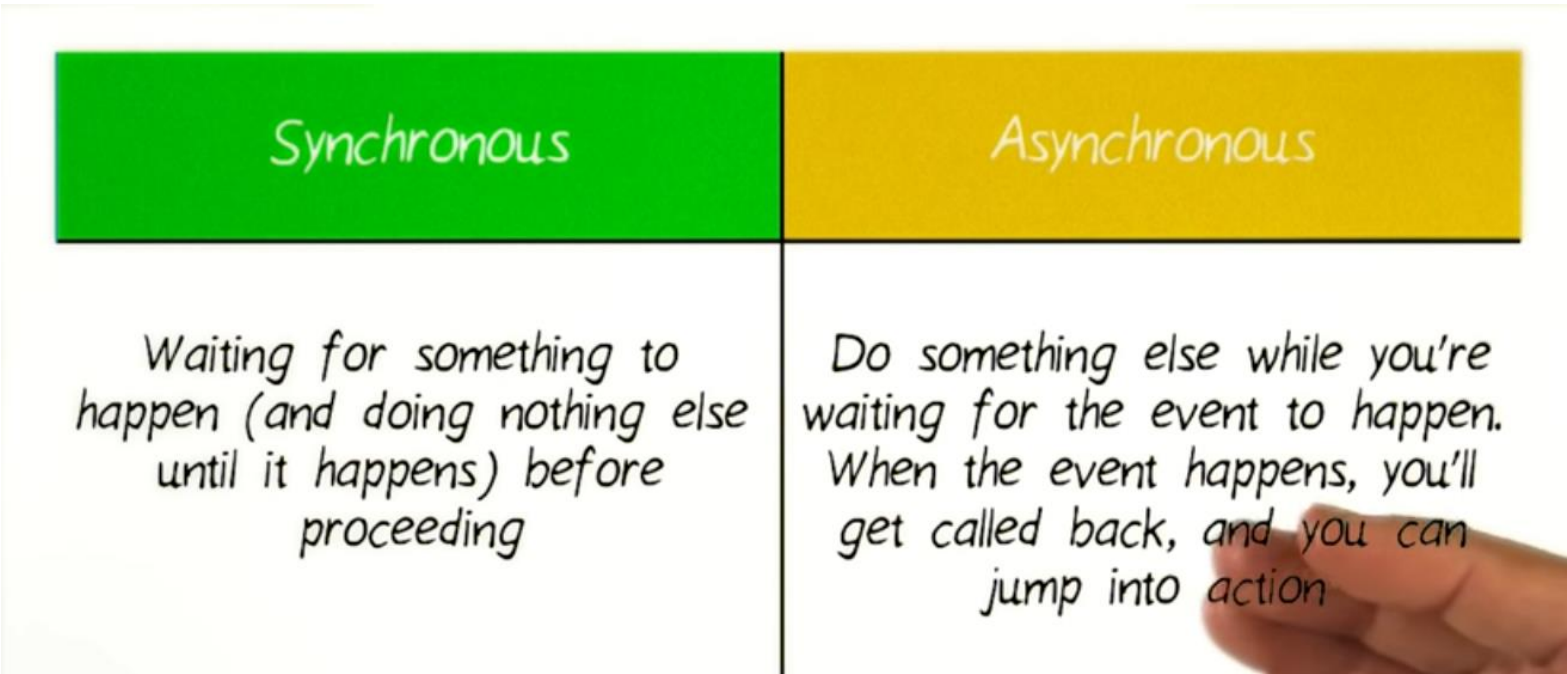
setOnClickListener Vs setOnItemClickListener // note difference between them

```
final TextView colorsTextView = (TextView) findViewById(R.id.colors);
colorsTextView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intentColor = new Intent(colorsTextView.getContext(),ColorsActivity.class);
        startActivity(intentColor);
    }
});
```

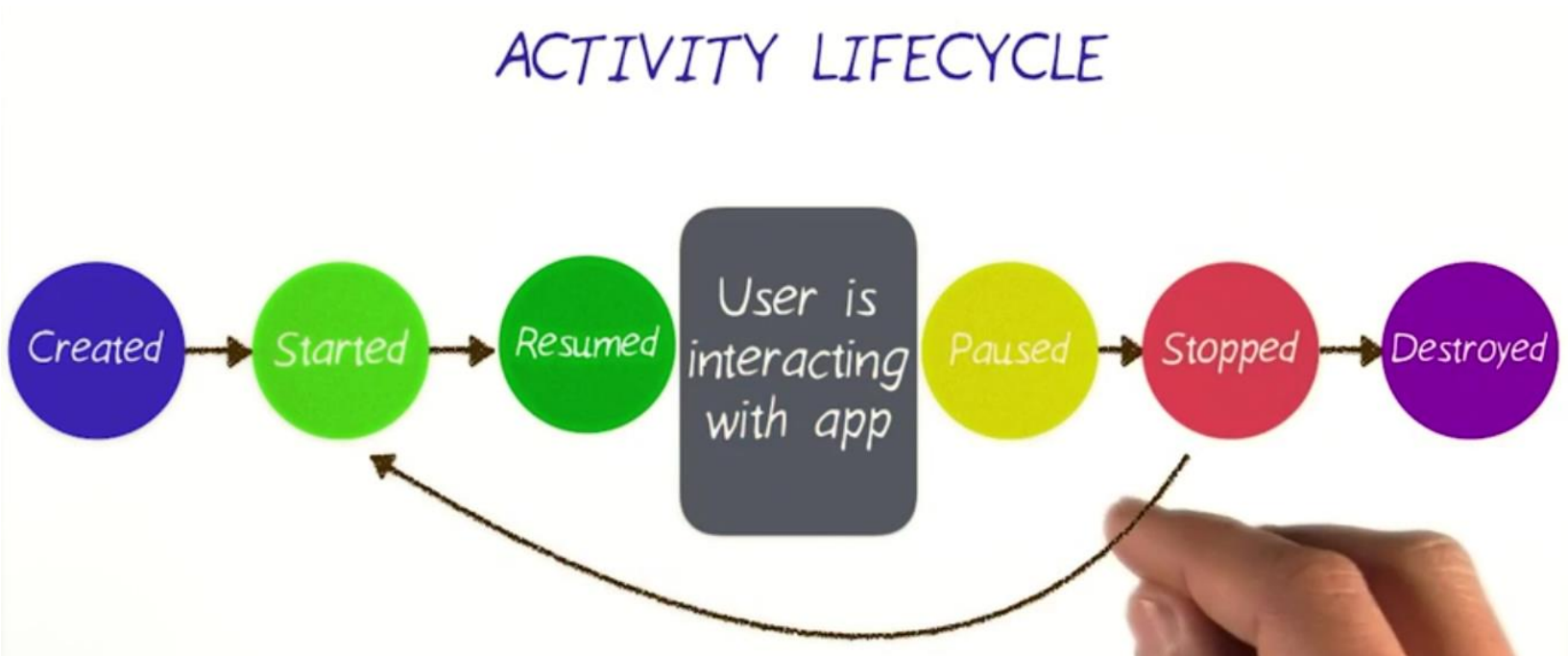
```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        Toast.makeText(getApplicationContext(),"Clicked",Toast.LENGTH_SHORT).show();
    }
});
```

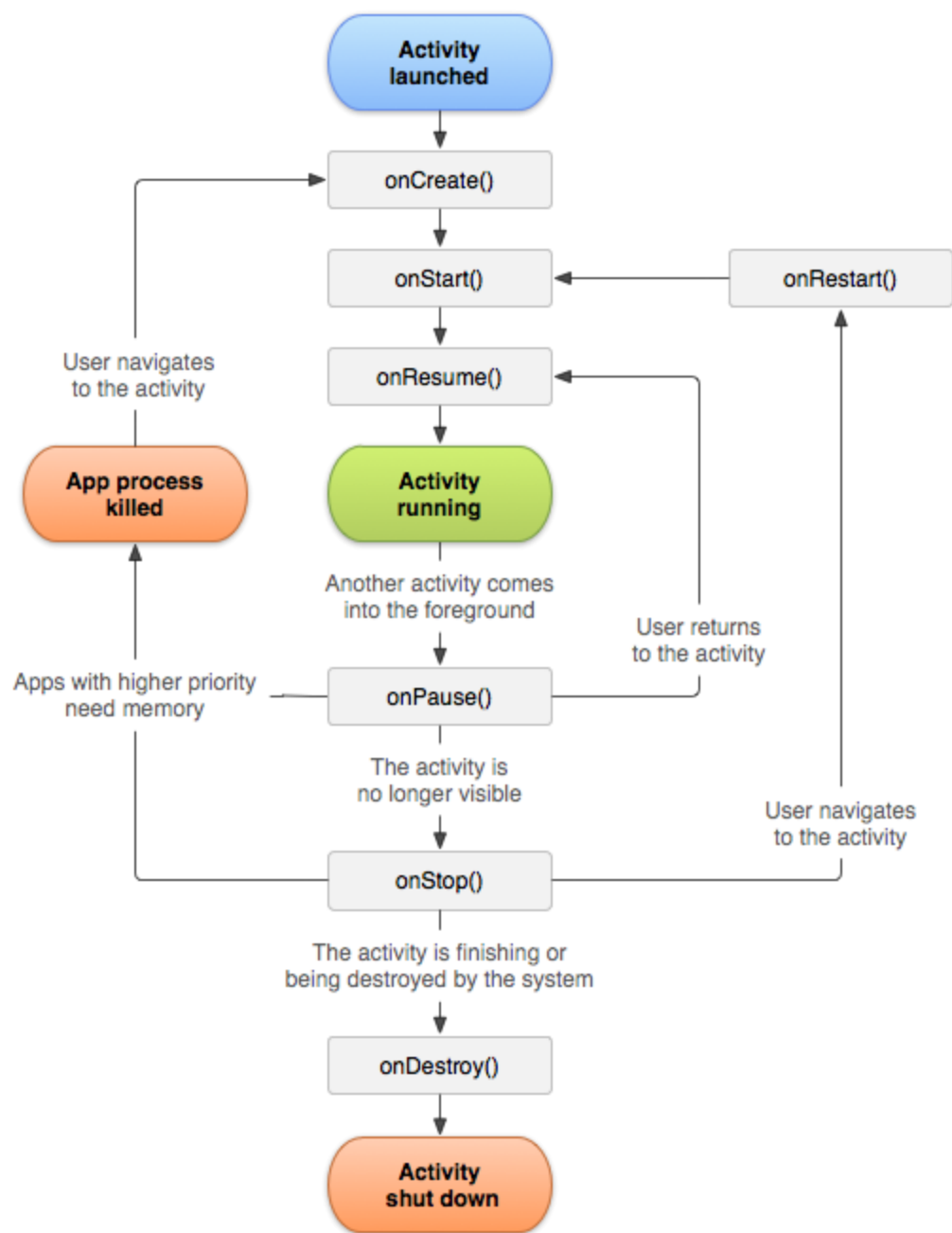
```
mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
    @Override
    public void onCompletion(MediaPlayer mediaPlayer) {
        Toast.makeText(getApplicationContext(),R.string.amen,Toast.LENGTH_SHORT).show();
    }
});
```

Synchronous vs Asynchronous (Async callback)
(Async callback) like onItemClickListener and OnClickListener



Android Activity Lifecycle



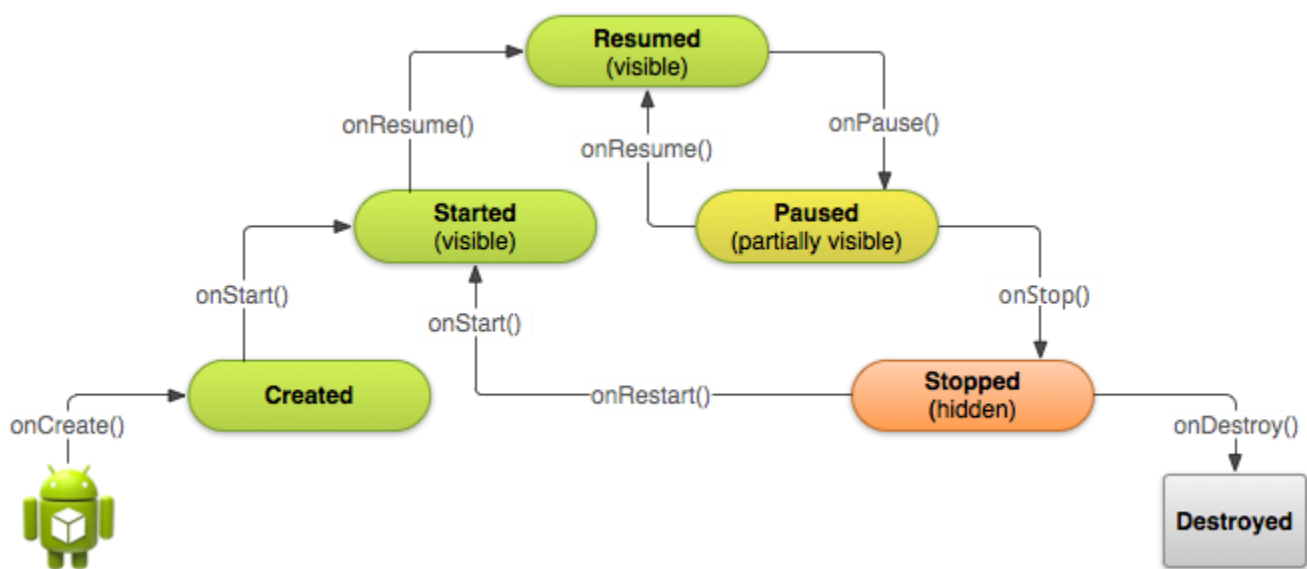


Activity has six states

- *Created*
- *Started*
- *Resumed*
- *Paused*
- *Stopped*
- *Destroyed*

Activity lifecycle has seven methods

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- **onRestart()**
- onDestroy()



Situations

- **When open the app**

onCreate() --> onStart() --> onResume()

- When back button pressed and exit the app

onPaused() -- > onStop() --> onDestory()

- When home button pressed

onPaused() --> onStop()

- After pressed home button when again open app from recent task list or clicked on icon

onRestart() --> onStart() --> onResume()

- When open app another app from notification bar or open settings

onPaused() --> onStop()

- Back button pressed from another app or settings then used can see our app

onRestart() --> onStart() --> onResume()

- When any dialog open on screen

onPause()

- After dismiss the dialog or back button from dialog

onResume()

- Any phone is ringing and user in the app

onPause() --> onResume()

- When user pressed phone's answer button

onPause()

- After call end

onResume()

- When phone screen off

onPaused() --> onStop()

- When screen is turned back on

onRestart() --> onStart() --> onResume()

Audio Focus States

AUDIO FOCUS STATES

Audio Focus State	Description of this state in your own words	Describe what we should do in the Miwok app when we enter this state
AUDIOFOCUS_GAIN	Gain audio focus back again (after having lost it earlier)	Resume playing the audio file
AUDIOFOCUS_LOSS	Permanent loss of audio focus	Stop the MediaPlayer and release resources
AUDIOFOCUS_LOSS_TRANSIENT	Temporary loss of audio focus	Pause audio file
AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK	Temporary loss of audio focus, can "duck" or lower volume if applicable	Pause audio file (each part of the word pronunciation is important to be heard)

In ListView you can put this to make feedback effect pressed state for ListItem

android:drawSelectorOnTop="true"

in View you can do that for touch feedback for clickable view in View

android:background="?android:attr/selectableItemBackground"