List , Intent and Activity Lifecycle

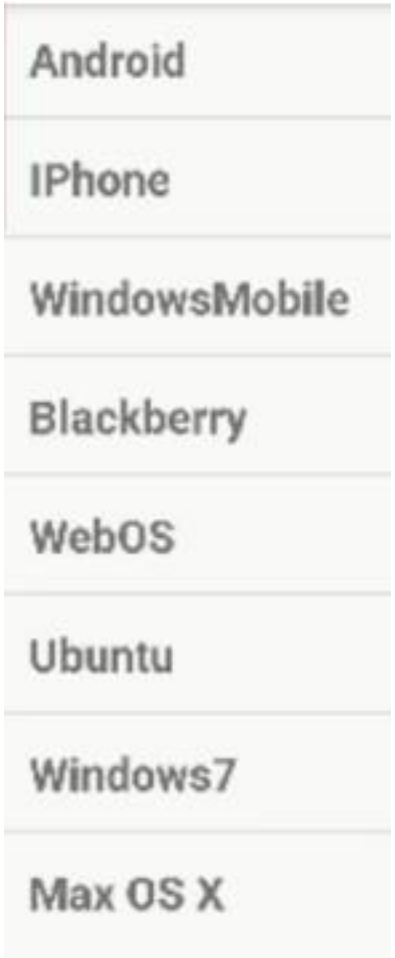
Android Application Development

ListView

Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

android:id="@+id/*theID*"	unique ID for use in Java code
android:clickable="*bool*"	set to `false` to disable the button
android:entries="@array/*array*"	set of options to appear in the list (must match an array in **strings.xml**)

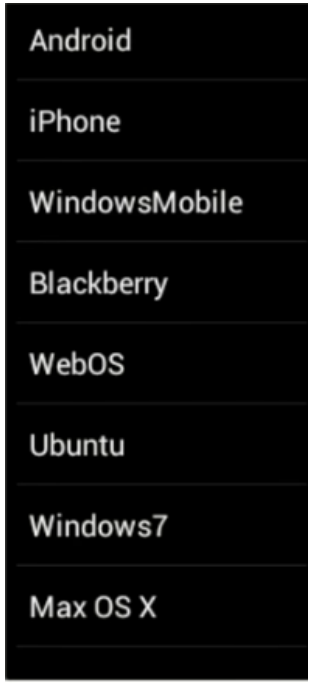
key attributes in XML



Types of List

- **Static List** : content is fixed and known before the app runs.
- *Declare the list elements in the string.xml resource file.*

- **Dynamic List** : Content is read or generated as the program runs.
- *Come from a data file, or from the internet , etc.*
- *Must be set in java code.*



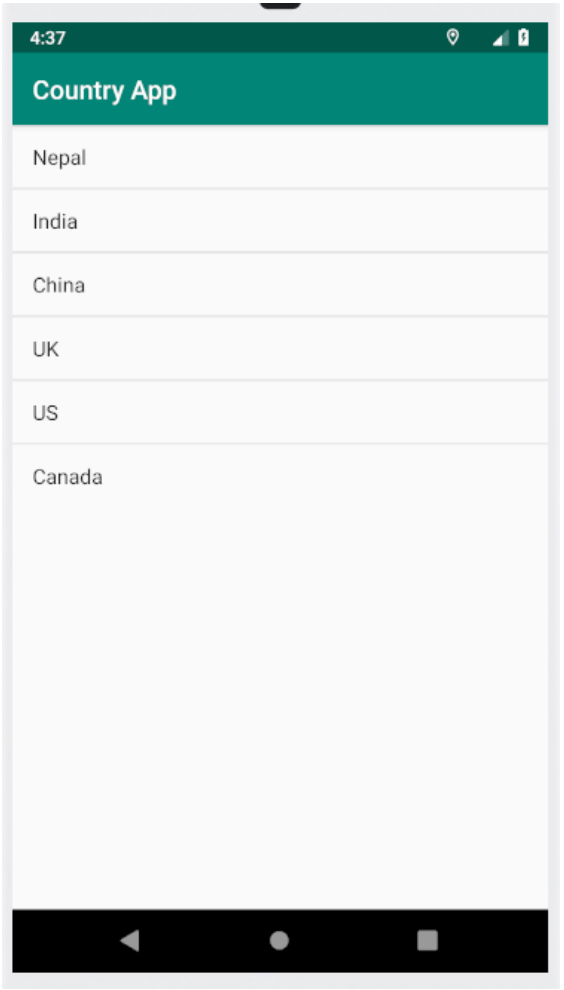
Static List : Example

string.xml

```
<resources>
    <string name="app_name">Dictionary</string>

    <string-array name="countries">
        <item>Nepal</item>
        <item>India</item>
        <item>China</item>
        <item>US</item>
        <item>UK</item>
        <item>Canada</item>
    </string-array>

</resources>
```

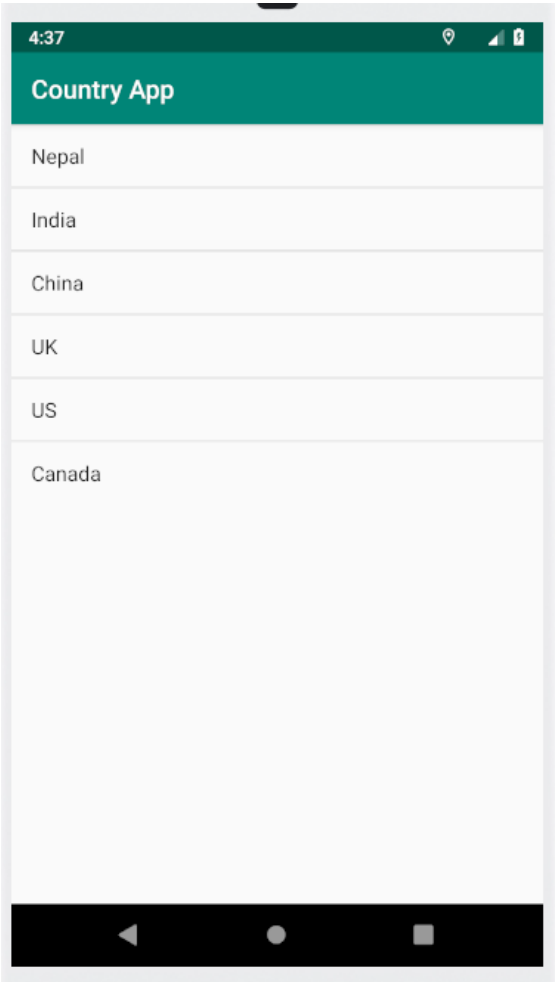
Layout.xml

```
<LinearLayout 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:orientation="vertical"
    tools:context=".DictionaryActivity">

    <ListView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:entries="@array/countries"
        >

    </ListView>
</LinearLayout>
```



Using java code

```
public class CountryActivity extends AppCompatActivity {

    public static final String countries[] = {
            "Nepal" ,
            "India" ,
            "China",
            "UK",
            "US",
            "Canada"
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_dictionary);
        ListView lstCountries = findViewById(R.id.lstCountries);

        ArrayAdapter countryAdapter = new ArrayAdapter<>(
                context: this, // Activity
               android.R.layout.simple_list_item_1,  // layout
                countries // array
        );

        lstCountries.setAdapter(countryAdapter);
    }
}
```



List Adapters

adapter : Helps turn list data into list view items.

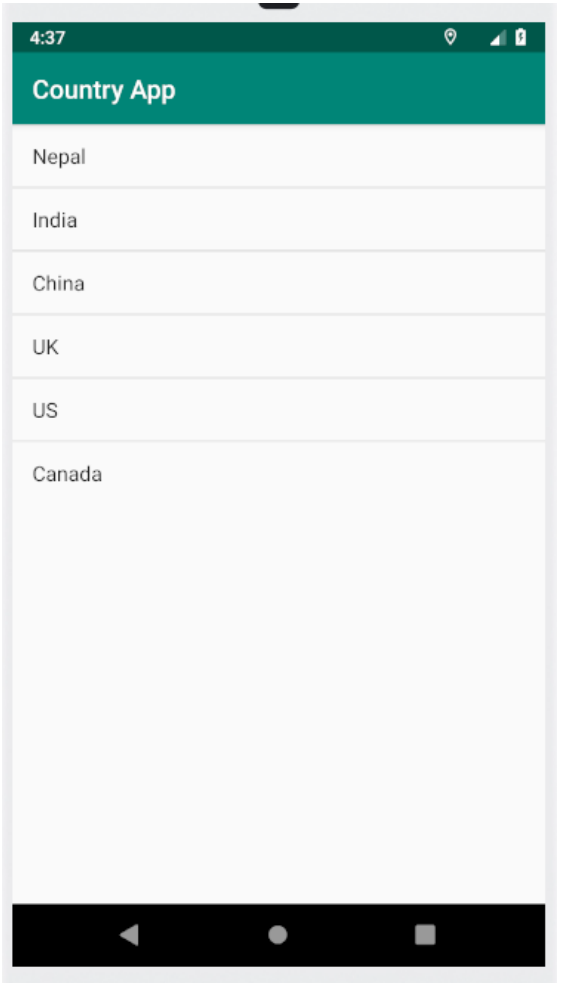
- **Common adapters**
- ArrayAdapter : items come from an array or list.
- CursorAdapter : items come from a database query

- Syntax for creating an adapter

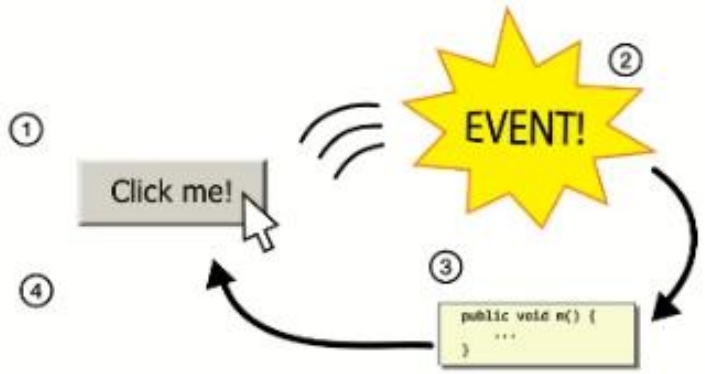
```
1  ArrayAdapter<String> name =
2          new ArrayAdapter<>(activity, layout, array);
```

Dictionary App

```
public class CountryActivity extends AppCompatActivity {

    public static final String countries[] = {
            "Nepal" , "Kathmandu",
            "India" , "New Delhi",
            "China", "Beijing",
            "UK", "London",
            "US","Washington, D.C.",
            "Canada","Ottawa"
    };
    private Map<String,String> dictionary;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_dictionary);
        ListView lstCountries = findViewById(R.id.lstCountries);

        dictionary =new HashMap<>();
        for(int i=0;i<countries.length;i+=2){
            dictionary.put(countries[i],countries[i+1]);
        }

        ArrayAdapter countryAdapter = new ArrayAdapter<>(
                context: this, // Activity
                android.R.layout.simple_list_item_1,  // layout
                new ArrayList<String>(dictionary.keySet()) //Arraylist key value
        );
        lstCountries.setAdapter(countryAdapter);
    }
}
```



Handling list events



- List don’t use the onClick event.
- Event listeners must be attached in java, not XML
- We must use Java anonymous inner classes.
- **anonymous inner classes :** A shorthand syntax for declaring a small class without giving it an explicit name. It is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain “extras” such as overloading methods of a class or interface, without having to actually subclass a class.

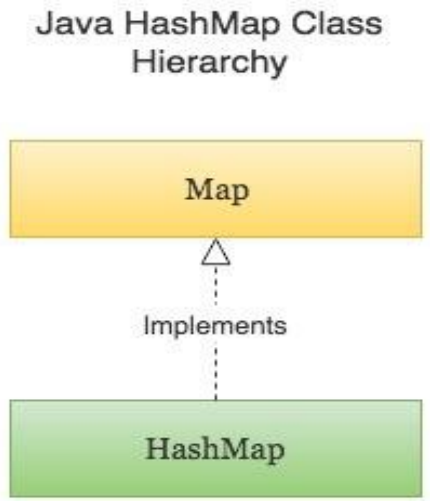
List events

- List views respond to the following events:
  - `setOnItemClickListener(AdapterView.OnItemClickListener)`
    Listener for when an item in the list has been clicked.
  - `setOnItemLongClickListener(AdapterView.OnItemLongClickListener)`
    Listener for when an item in the list has been clicked and held.
  - `setOnItemSelectedListener(AdapterView.OnItemSelectedListener)`
    Listener for when an item in the list has been selected.
  - Others: `onDrag`, `onFocusChanged`, `onHover` `onKey`, `onScroll`, `onTouch`, ...

HashMap

- Java HashMap is a hash table based implementation of Java's Map interface. A Map, as you might know, is a collection of key-value pairs. It maps keys to values.

- Following are few key points to note about HashMap in Java –
- A HashMap cannot contain duplicate keys.
- Java HashMap allows one **null** and the **null** key.
- HashMap is an unordered collection. It does not guarantee any specific order of the elements.



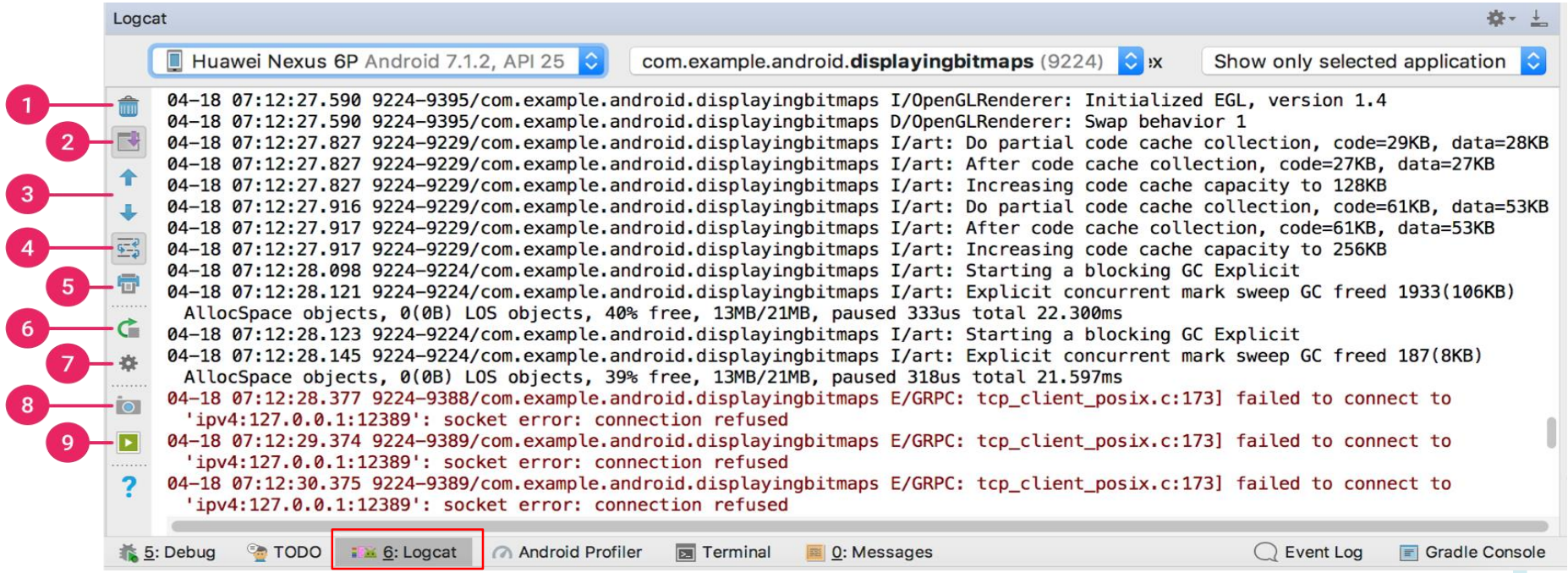
HashMap - Example

```
public class HashMapActivity extends AppCompatActivity {
    private static final String TAG = HashMapActivity.class.getSimpleName();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hash_map);
        // Create a HashMap object called capitalCities
        HashMap<String,String> capitalCities = new HashMap<>();
        // Add keys and values (Country, Capital)
        capitalCities.put("Nepal","Kathmandu");
        capitalCities.put("India" , "New Delhi");
        capitalCities.put("China" , "Beijing");
        capitalCities.put("UK" , "London");
        // Display all the map key and value in logcat
        Log.d(TAG, msg: "Countries and Capitals : " + capitalCities);
    }
}
```

Android LOG

The **Logcat** window in Android Studio displays system messages, such as when a garbage collection occurs, and messages that you added to your app with the Log class.

It displays messages in real time and keeps a history so you can view older messages.



LOG Methods

Log.e :

The e in Log.e stands for **error**. Use this when you *know* an error has occurred, and you're logging details about that error.

Log.w :

The w in Log.w stands for **warning**. Use this when you suspect an issue may be occurring, but haven't yet received full-on error messages.

Log.i :

The i in Log.i stands for **information**. Use this to post useful information. For instance, maybe you want to double-check a method is being called successfully, you could print an informational message to the log reading something like *"X method called!"*.

LOG Methods - 2

Log.d :

The d in Log.d stands for **debug**. As you might imagine, you'll use this one for debugging purposes. You'll probably use this one most frequently out of all available log methods.

Log.v :

The v in Log.v stands for **verbose**. Use this when you're implementing many, many different log statements as a debugging approach.

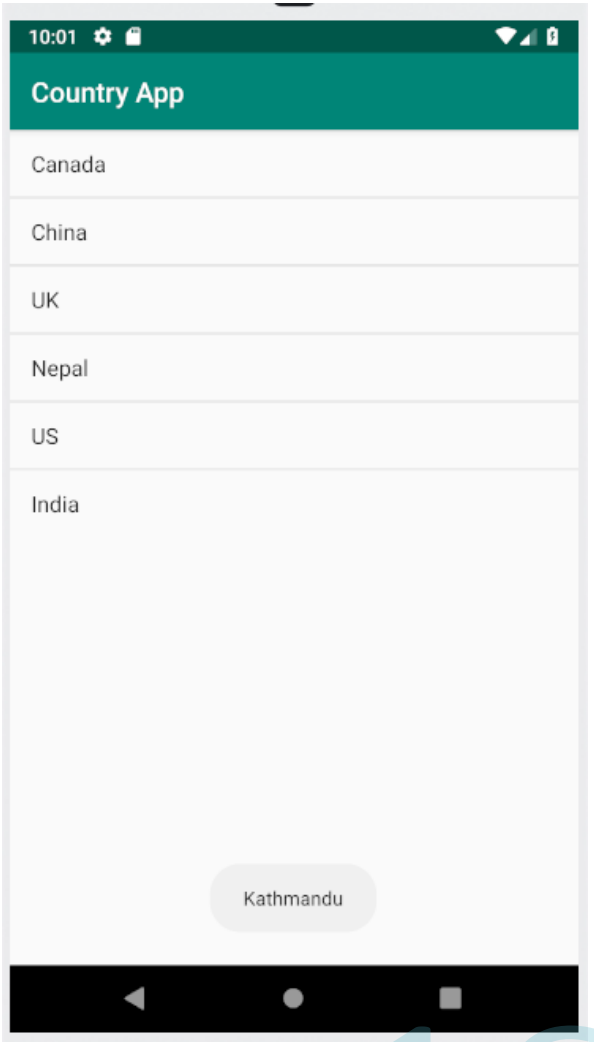
Logcat output : using log.d

```
1 11:26:11.220 21075-21075/com.dictionaryapp I/InstantRun: starting instant run server: is main process
1 11:26:11.430 21075-21075/com.dictionaryapp W/m.dictionaryap: Accessing hidden method Landroid/view/View;->computeFitSystemWindows(Landroid/graphics/Re
1 11:26:11.431 21075-21075/com.dictionaryapp W/m.dictionaryap: Accessing hidden method Landroid/view/ViewGroup;->makeOptionalFitsSystemWindows()V (light
1 11:26:11.481 21075-21075/com.dictionaryapp D/HashMapActivity: Countries and Capitals : {China=Beijing, UK=London, Nepal=Kathmandu, India=New Delhi}
1 11:26:11.498 21075-21075/com.dictionaryapp D/OpenGLRenderer: Skia GL Pipeline
1 11:26:11.549 21075-21099/com.dictionaryapp I/ConfigStore: android::hardware::configstore::V1_0::ISurfaceFlingerConfigs::hasWideColorDisplay retrieved:
1 11:26:11.549 21075-21099/com.dictionaryapp I/ConfigStore: android::hardware::configstore::V1_0::ISurfaceFlingerConfigs::hasHDRDisplay retrieved: 0
1 11:26:11.549 21075-21099/com.dictionaryapp I/OpenGLRenderer: Initialized EGL, version 1.4
1 11:26:11.549 21075-21099/com.dictionaryapp D/OpenGLRenderer: Swap behavior 1
```

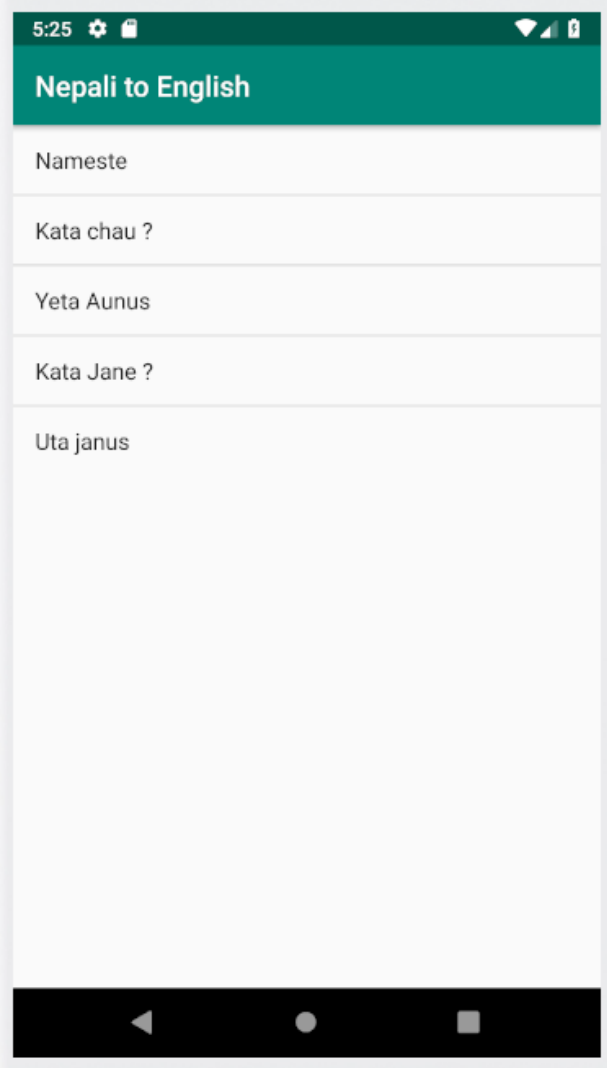
Dictionary App

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_dictionary);
    ListView lstCountries = findViewById(R.id.lstCountries);

    dictionary =new HashMap<>();
    for(int i=0;i<countries.length;i+=2){
        dictionary.put(countries[i],countries[i+1]);
    }

    ArrayAdapter countryAdapter = new ArrayAdapter<>(
            context: this, // Activity
            android.R.layout.simple_list_item_1,  // layout
           new ArrayList<String>(dictionary.keySet()) //Arraylist key value
    );
    lstCountries.setAdapter(countryAdapter);

    lstCountries.setOnItemClickListener((parent, view, position, id) → {

            String country = parent.getItemAtPosition(position).toString();
            String capital = dictionary.get(country);
            Toast.makeText(getApplicationContext(),capital.toString(),Toast.LENGTH_LONG).show();
    });
}
```

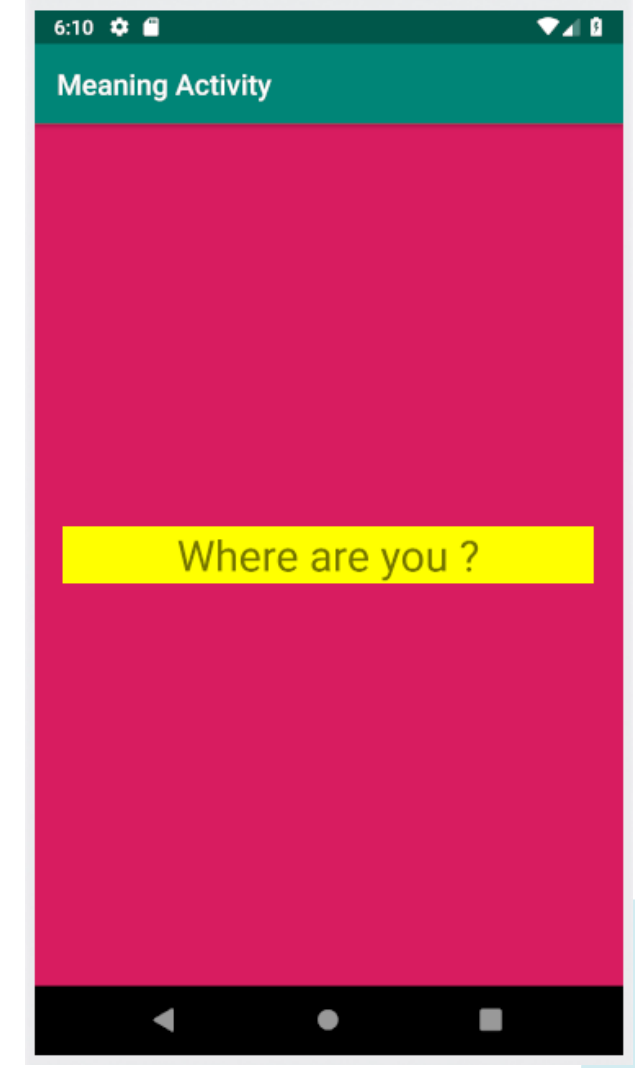
Getting the current item from the current position



Another Dictonary App

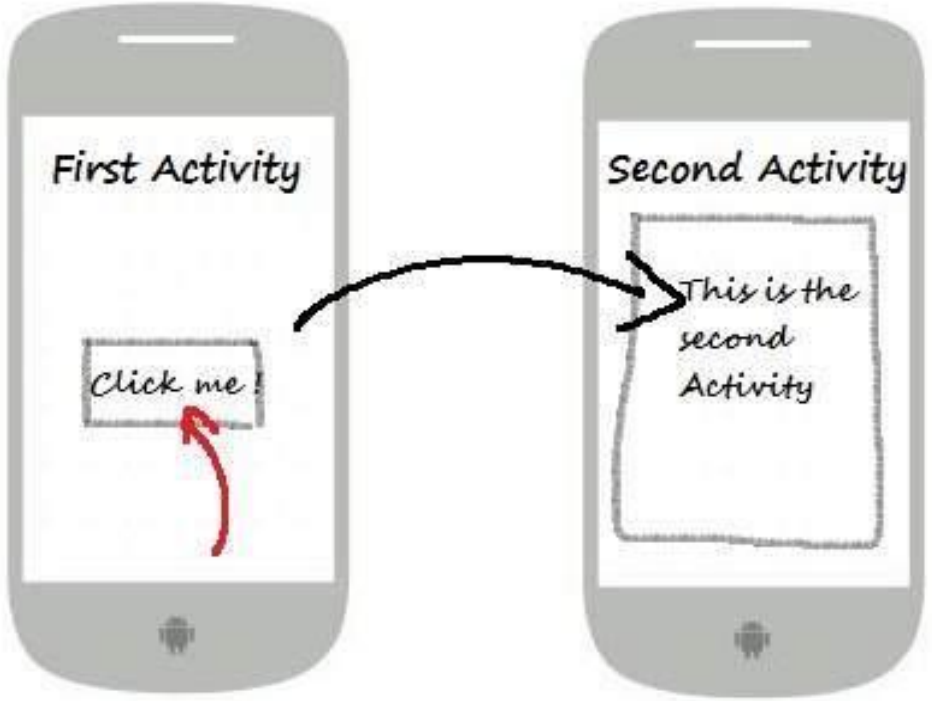


Meaning will be display in another activity



Intent

- An Intent is a simple message object that is used to communicate between android components such as activities, content providers, broadcast receivers and services.
- Intents are also used to **transfer** data between activities.
- Intents are used generally for **starting** a new activity using startActivity().



Types of Intent

1. Implicit Intent
2. Explicit Intent

Implicit Intent

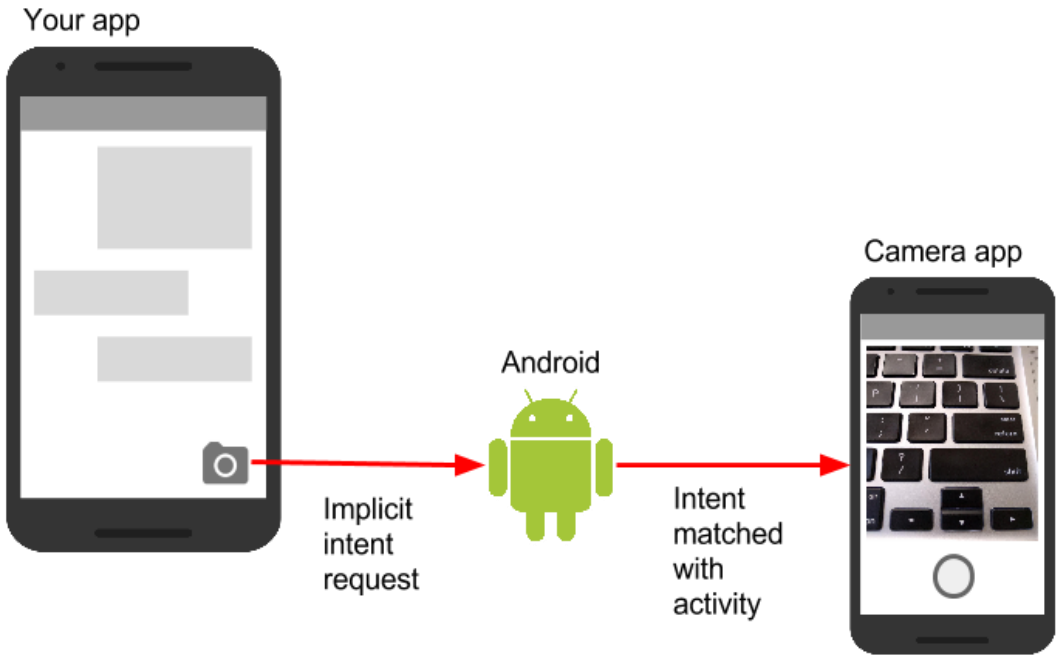
- The implicit intent is the intent where instead of defining the exact components, you define the action that you want to perform for different activities.
- An Implicit intent specifies an action that can invoke any app on the device to be able to perform an action.
- Using an Implicit Intent is useful when your app cannot perform the action but other apps probably can and you'd like the user to pick which app to use.

  Implicit Intents do not directly specify the Android components which should be called, it only specifies action to be performed. A Uri can be used with the implicit intent to specify the data type.

  for example

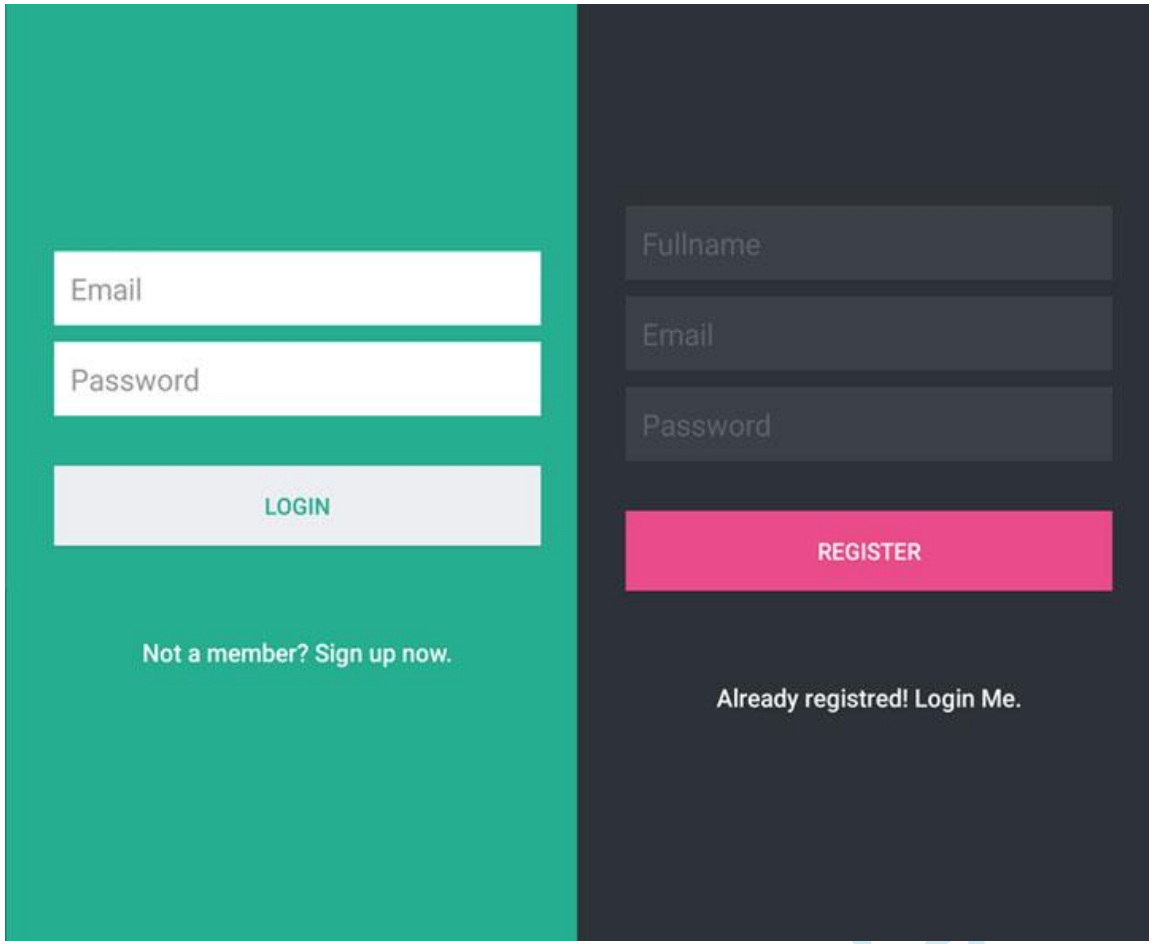
  Intent intent = new Intent(ACTION_VIEW,Uri.parse("http://www.google.com"));

  this will cause web browser to open a webpage. Android system searches for all components which are registered for the specific action and the data type.If many components are found then the user can select which component to use.

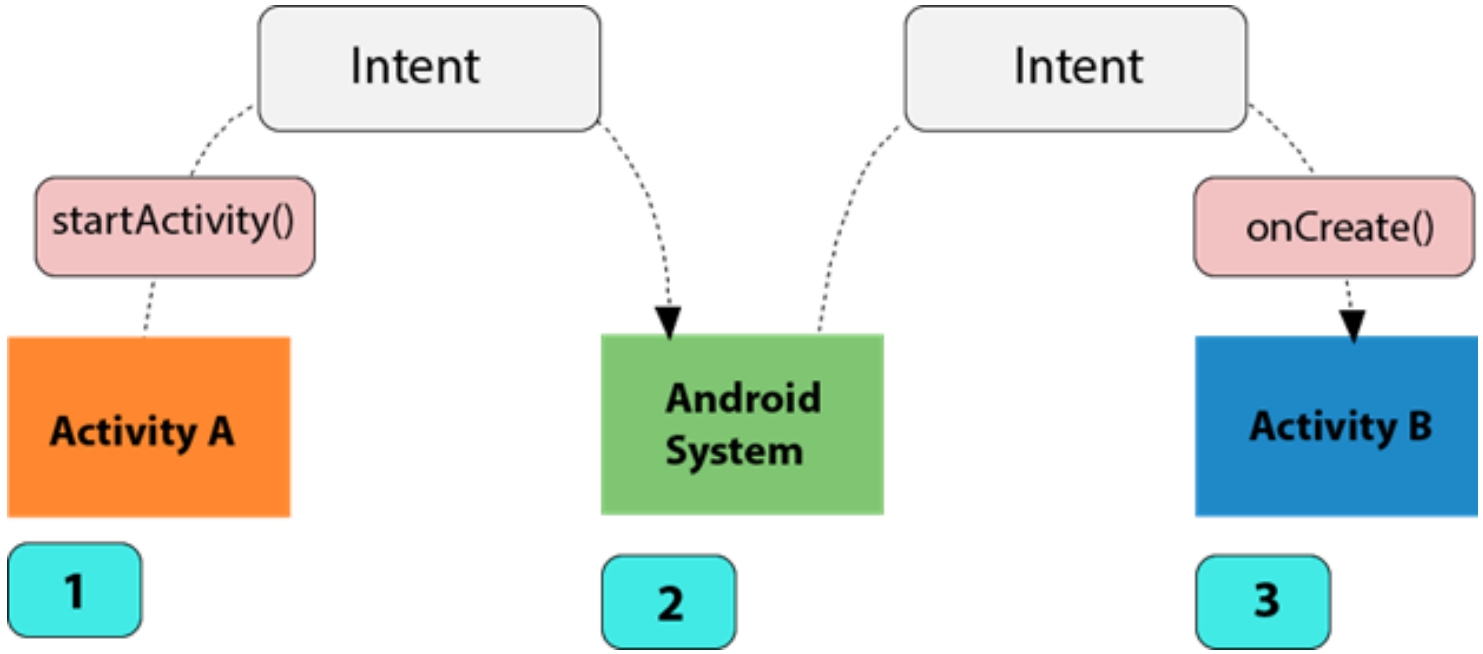


Explicit Intent

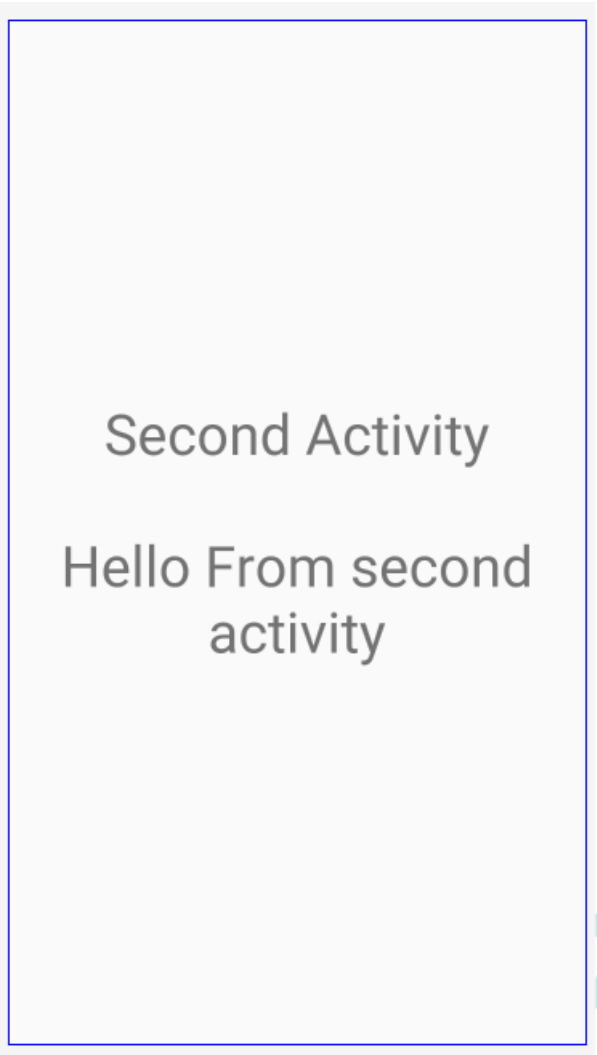
- An explicit intent is an Intent where you explicitly define the component that needs to be called by the Android System.
- An explicit intent is one that you can use to launch a specific app component, such as a particular activity or service in your app.
- Explicit intents are used in the application itself wherein one activity can switch to other activity...Example Intent intent = new Intent(this,Target.class); this causes switching of activity from current context to the target activity. Explicit Intents can also be used to pass data to other activity using putExtra method and retrieved by target activity by getIntent().getExtras() methods.



Connecting activities with intent



Sample Program



First Activity Code

```
public class IntentActivity extends AppCompatActivity {

    private Button btnOpen;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_intent);

        btnOpen = findViewById(R.id.btnOpen);

        btnOpen.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent( packageContext: IntentActivity.this,AnotherActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

Source

Destination

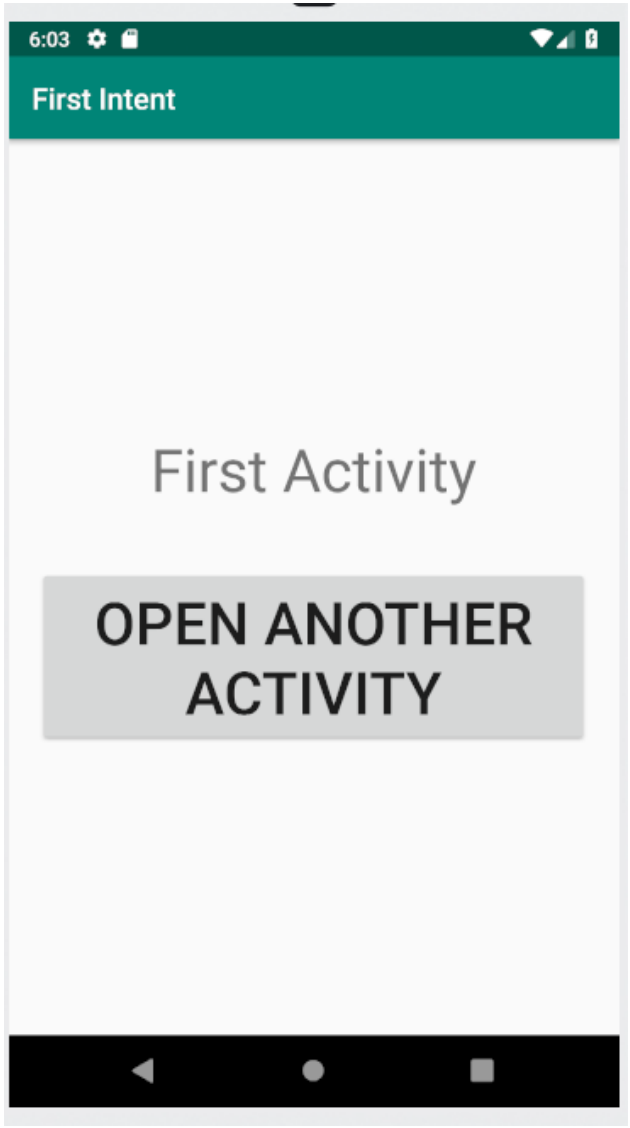
Sending message from one activity to another

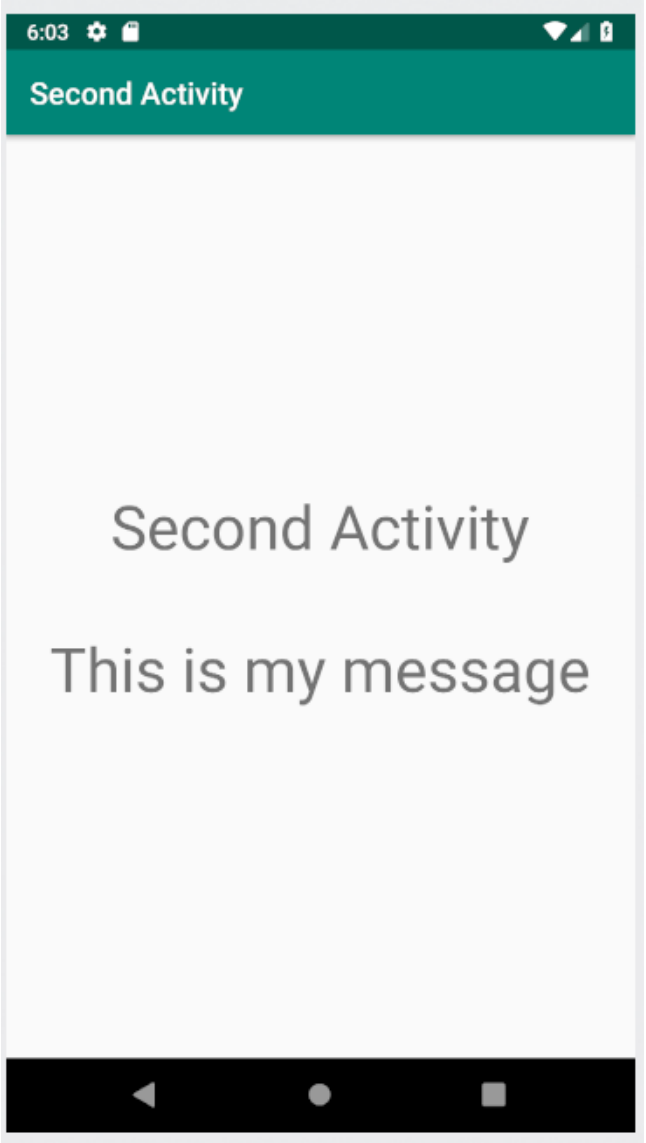
```
btnOpen.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent( packageContext: IntentActivity.this,AnotherActivity.class);
        intent.putExtra( name: "myMessage", value: "This is my message");
        startActivity(intent);
    }
});
```

Receiving message in another activity

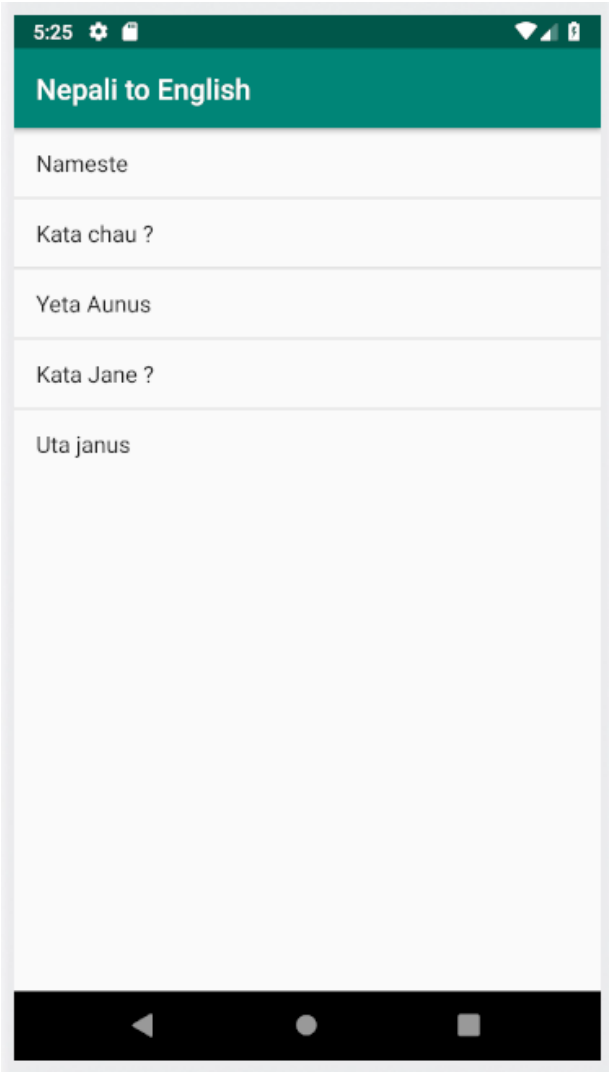
```
public class AnotherActivity extends AppCompatActivity {

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

        TextView tvText = findViewById(R.id.tvText);
        Bundle bundle = getIntent().getExtras();
        if(bundle!=null) {
            String message = bundle.getString( key: "myMessage");
            tvText.setText(message);
        }
        else
        {
            Toast.makeText( context: this,  text: "No Message", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Output

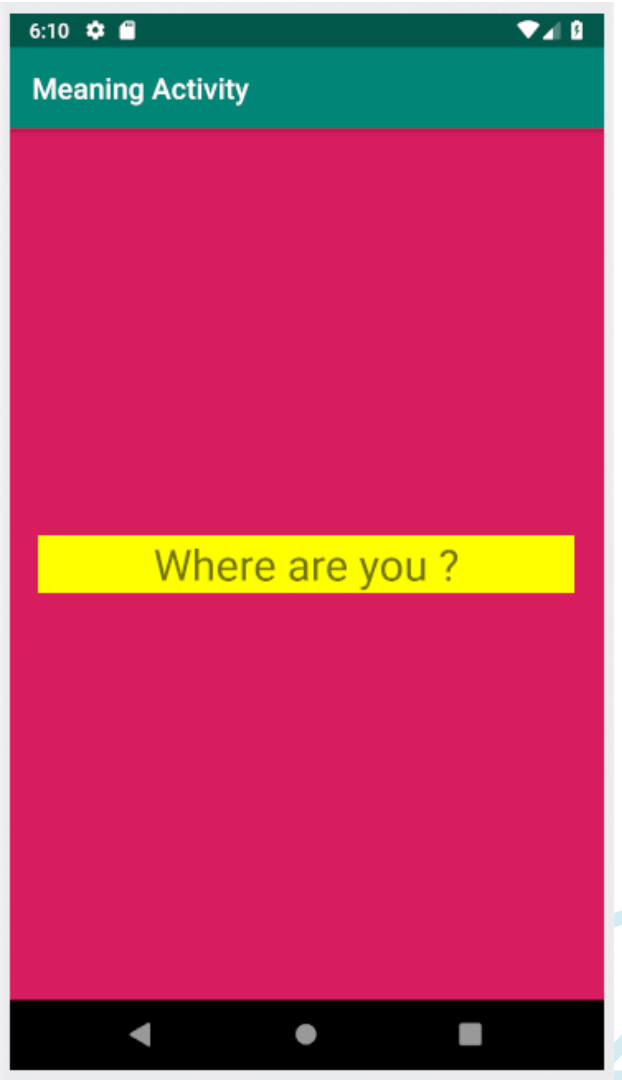

6:03
First Intent
First Activity
OPEN ANOTHER ACTIVITY


6:03
Second Activity
Second Activity
This is my message

Back To Dictonary App



Meaning will be display in another activity



First Activity

```
private ListView lstDictionary;
private Map<String, String> dictionary;

public static final String words[] = {
        "Yeta Aunus", "Come here",
        "Uta janus", "Go there",
        "Nameste", "Hello",
        "Kata chau ?", "Where are you ?",
        "Kata Jane ?", "Where are you going?"
};
```

```java
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_dictionary);

    lstDictionary = findViewById(R.id.lstDictionary);
    dictionary = new HashMap<>();

    for (int i = 0; i < words.length; i += 2) {
        dictionary.put(words[i], words[i + 1]); // putting key and value in hashmap
    }

    ArrayAdapter adapter = new ArrayAdapter<>(
            context: this,
            android.R.layout.simple_list_item_1,
            new ArrayList<String>(dictionary.keySet())
    );
    lstDictionary.setAdapter(adapter);

    lstDictionary.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            String key = parent.getItemAtPosition(position).toString(); // Get the current position
            String meaning = dictionary.get(key);   // Get the meaning of current position key

            // Intent will call MeaningActivity from DictionaryActivity
            Intent intent = new Intent( packageContext: DictionaryActivity.this,MeaningActivity.class);
            // We have to pass the message from this activity to MeaningActivity
            intent.putExtra( name: "meaning",meaning);
            startActivity(intent);
        }
    });
}
```

Another Activity

```
public class MeaningActivity extends AppCompatActivity {

    private TextView tvMeaning;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_meaning);

        Bundle bundle = getIntent().getExtras();

        if(bundle!=null) {
            String meaning = bundle.getString( key: "meaning");
            tvMeaning = findViewById(R.id.tvMeaning);
            tvMeaning.setText(meaning);
        }
        else
        {
            Toast.makeText( context: this,  text: "No Meaning", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Android Activity

An Android activity is one screen of the Android app's user interface. In that way an Android activity is very similar to windows in a desktop application. An Android app may contain one or more activities, meaning one or more screens. The Android app starts by showing the main activity, and from there the app may make it possible to open additional activities.

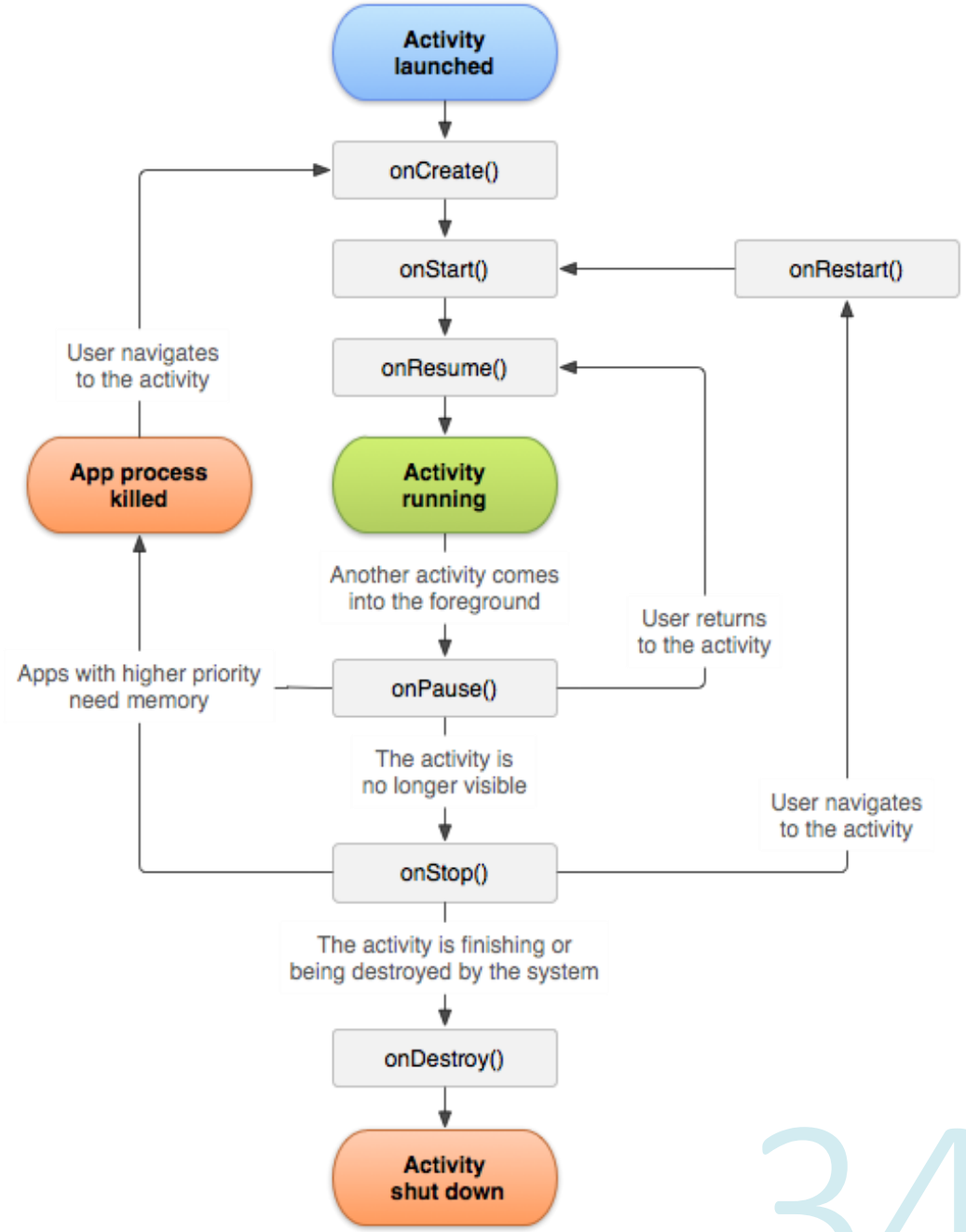
Declaring Activities in Manifest

<activity android:name=".SecondActivity"></activity>

<activity android:name=".ThirdActivity"></activity>

“SecondActivity” and “ThirdActivity” are the java class names.

Activity LifeCycle



Activity lifecycle methods

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.

Understanding Activity Life Cycle

As you have seen, an activity is simply a screen or user interface in an Android application—either a full screen or a floating window that a user interacts with. An Android app is made up of different activities that interact with the user as well as one another. For example, a simple calculator would use one single activity. If you enhanced the calculator app to switch between a simple version and a scientific version, you would then use two activities.

Every Android application runs inside its own process. Processes are started and stopped to run an application and also can be killed to conserve memory and resources. Activities, in turn, are run inside the main UI thread of the application's process.

Once an activity is launched, it goes through a **lifecycle**, a term that refers to the steps the activity progresses through as the user (and OS) interacts with it. There are specific method callbacks that let you react to the changes during the activity lifecycle.

The activity lifecyle has four states.

When the activity is on the foreground of the application, it is the *running* activity. Only one activity can be in the running state at a given time.

If the activity loses focus but remains visible (because a smaller activity appears on top), the activity is *paused*.

If the activity is completely covered by another running activity, the original activity is *stopped*. When an activity stops, you will lose any state and will need to re-create the current state of the user interface when the activity is restarted.

While the activity is paused or stopped, the system can kill it if it needs to reclaim memory. The user can restart the activity.

Understanding Activity Life Cycle

While the application moves through the different states, the android.app.Activity lifecycle methods (or callbacks) get called by the system. These callbacks are as follows.

onCreate(Bundle savedInstanceState)is called when the activity is created for the first time. You should initialize data, create an initial view, or reclaim the activity's frozen state if previously saved The onCreate callback is always followed by onStart.

onStart() is called when the activity is becoming visible. This is an ideal place to write code that affects the UI of the application, such as an event that deals with user interaction. This callback is normally followed by onResume but could be followed by onStop if the activity becomes hidden.

onResume() is called when the activity is running in the foreground and the user can interact with it. It is followed by onPause.

onPause() is called when another activity comes to the foreground. The implementation needs to be quick, because the other activity cannot run until this method returns. The onPause callback is followed by onResume if the activity returns to the foreground, or by onStop if the activity becomes invisible.

onStop() is called when the activity is invisible to the user; either a new activity has started, an existing activity has resumed, or this activity is getting destroyed. The onStop callback is followed by onRestart if the activity returns to the foreground.

onRestart() is called when the activity is being restarted, as when the activity is returning to the foreground. It is always followed by onStart.

onDestroy() is called by the system before the activity is destroyed, either because the activity is finishing or because the system is reclaiming the memory the activity is using.

Understanding Activity Life Cycle

Now let's look at how the Android activity lifecycle works. you overrode the onCreate method before. Now you'll override the remaining lifecycle methods in your mainActivity class by following these steps.

Open the **mainActivity.java** file in the project, and override the existing onStart method, which is called when the activity is first viewed. Call the onStart method of the parent class, and log a debug message:

@Override **public void** onStart(){ **super.**onStart(); Log.*d*(*CLASS_NAME*, "onStart"); }

Override the existing onPause method, which is called when another activity is called to the foreground. Call the onPause method of the parent and log a debug message:

@Override **public void** onPause(){ **super.**onPause(); Log.*d*(*CLASS_NAME*, "onPause"); }

Override the existing onResume method, which is called when the activity is running in the foreground and the user can interact with it. Call the onResume method of the parent class, and log a debug message:

@Override **public void** onResume(){ **super**.onResume(); Log.*d*(*CLASS_NAME*, "onResume"); }

Override the existing onStop method, which is called when the activity is invisible to the end user. Call the onStop method of the parent class, and log a debug message:

@Override **public void** onStop(){ **super.**onStop(); Log.*d*(*CLASS_NAME*, "onStop"); }

Understanding Activity Life Cycle

Override the existing onDestroy method, which is called when the activity is removed from the system and can no longer be interacted with. Call the onDestroy method of the parent class, and log a debug message:

@Override **public void** onDestroy(){ **super**.onDestroy(); Log.*d*(*CLASS_NAME*, "onDestroy"); }

Override the existing onRestart method, which is called when the activity is started again and returns to the foreground. Call the onRestart method of the parent class and log a debug message:

@Override **public void** onRestart(){ **super.**onRestart(); Log.*d*(*CLASS_NAME*, "onRestart"); }

Now **debug** your application on a device, and look at the debug messages (in the *LogCat* view) that show the changes of state in the application.