Mobile Application Development

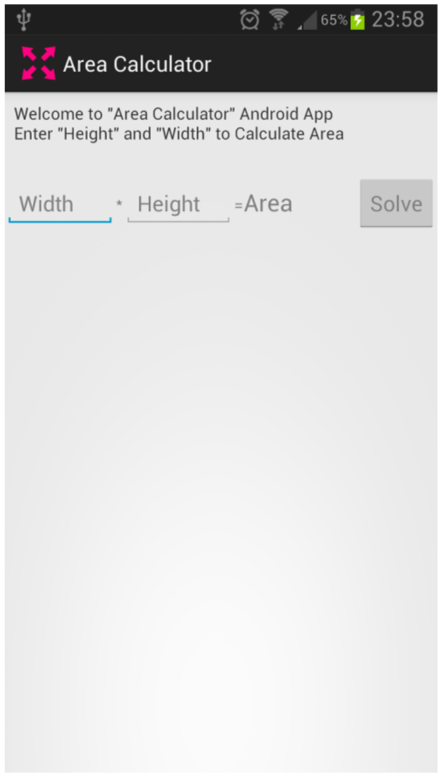
Activities & Intents

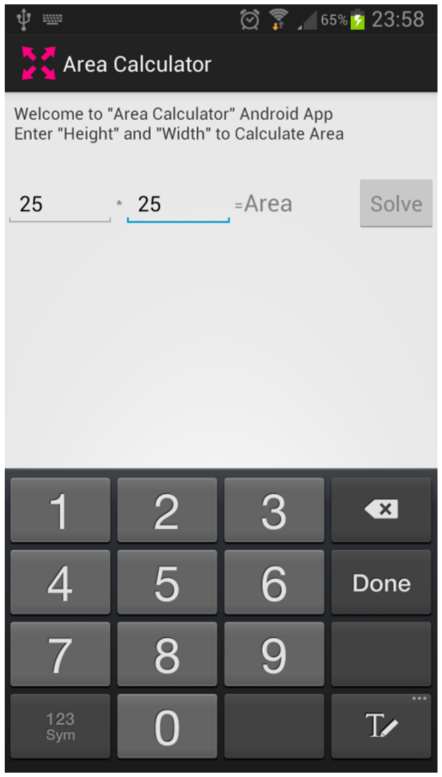
Activity

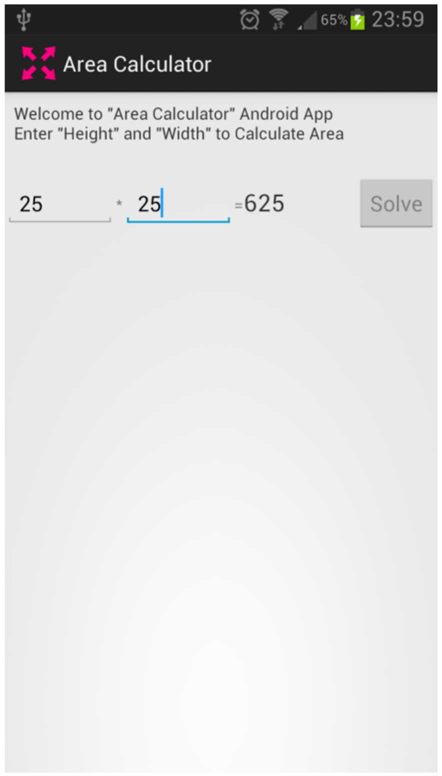
- An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.
- An application usually consists of multiple activities that are loosely bound to each other.
- Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time.
- Each activity can then start another activity in order to perform different actions.

Activity

- An *activity* represents a single screen with a user interface.


23:58
Area Calculator
Welcome to "Area Calculator" Android App
Enter "Height" and "Width" to Calculate Area
Width
*
Height
=Area
Solve

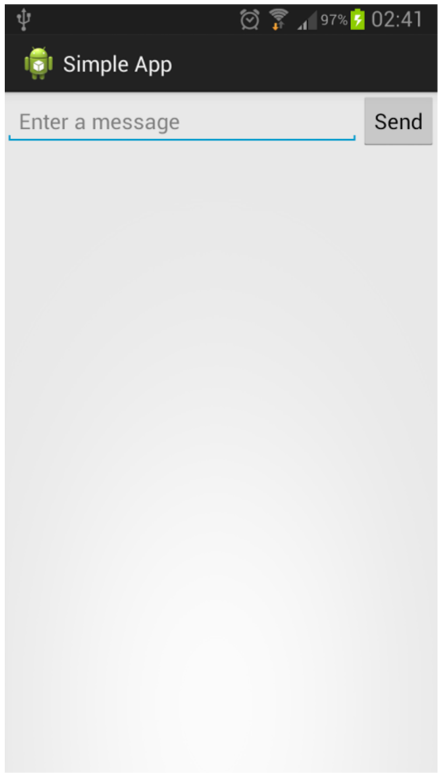

23:58
Area Calculator
Welcome to "Area Calculator" Android App
Enter "Height" and "Width" to Calculate Area
25
*
25
=Area
Solve
1
2
3
4
5
6
Done
7
8
9
123 Sym
0

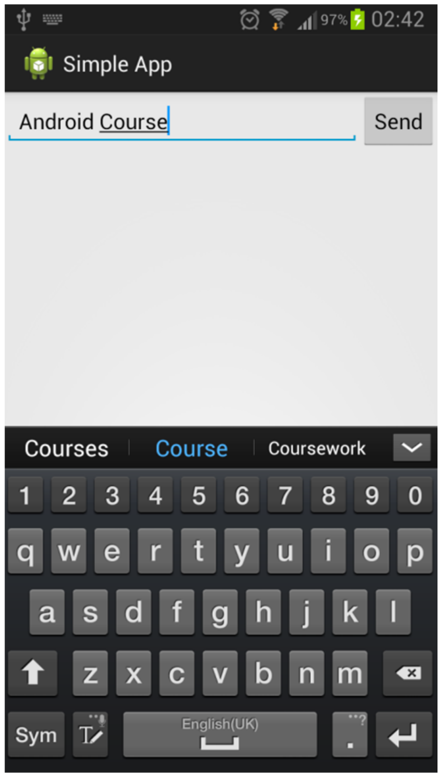

23:59
Area Calculator
Welcome to "Area Calculator" Android App
Enter "Height" and "Width" to Calculate Area
25
*
25
=625
Solve

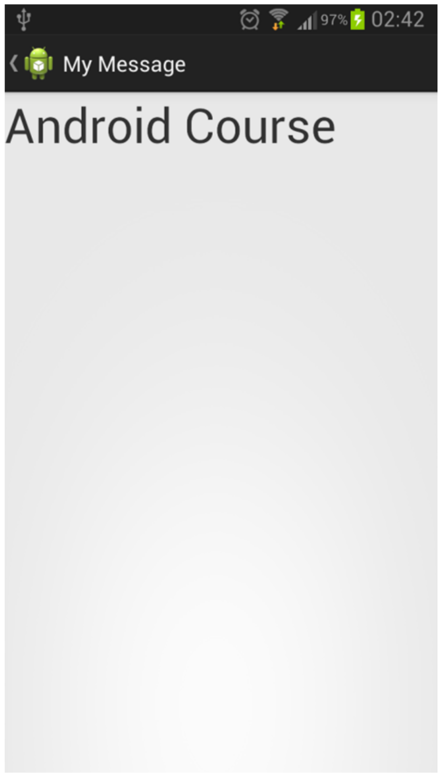
ACTIVITY
LIFECYCLE STATES & CALLBACKS

Activity

- An app might have more than one activity.







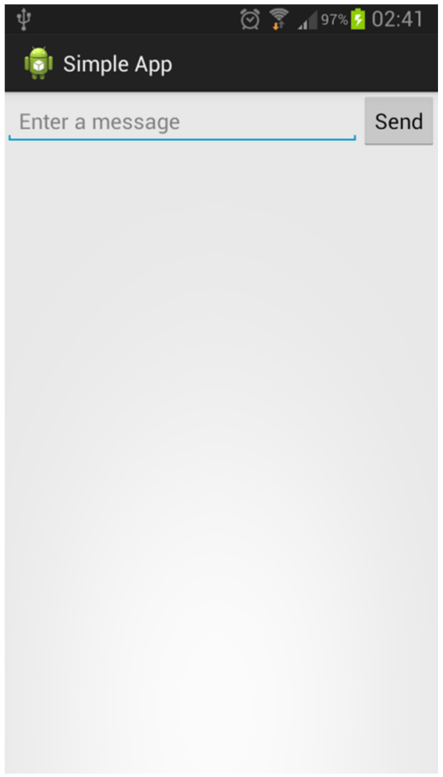
Android Course

Activity

- Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack").
- When a new activity starts, it is pushed onto the back stack and takes user focus.
- The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses the Back button, it is popped from the stack (and destroyed) and the previous activity resumes.

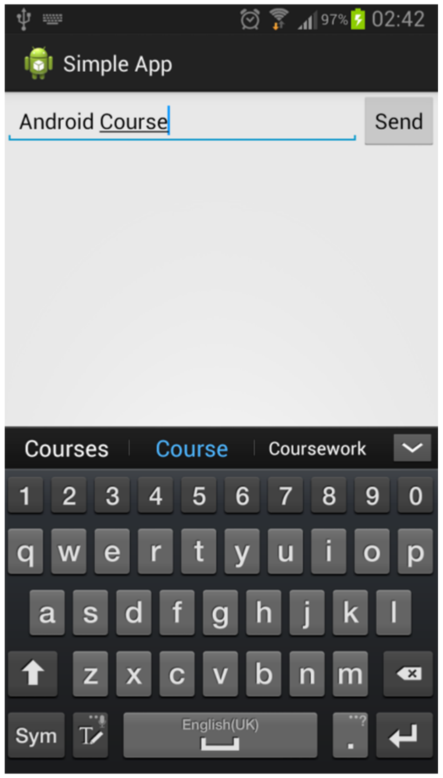
Back Stack

- The “main” Activity.



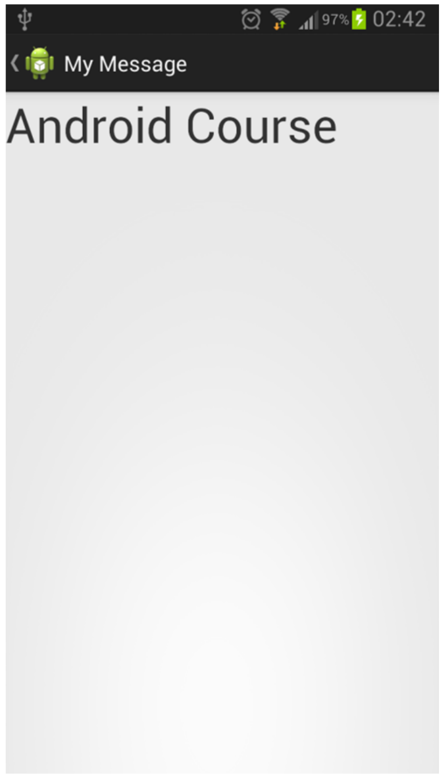
Back Stack

- The “main” Activity.



Back Stack

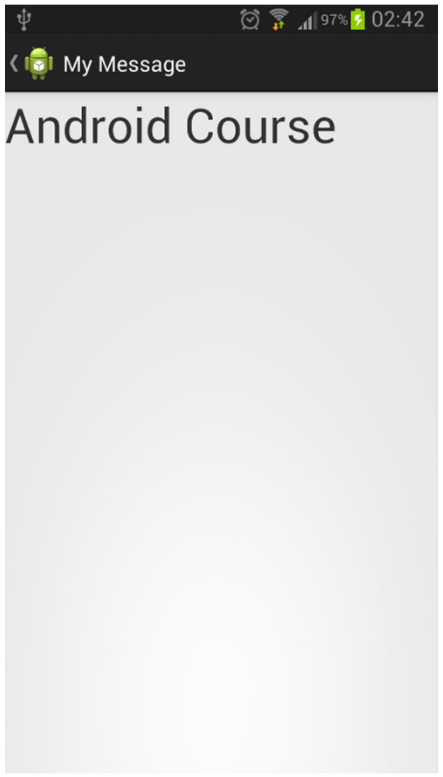
- The "main" Activity.
- "New" Activity is pushed onto the back stack and takes user focus.



Android Course

Back Stack

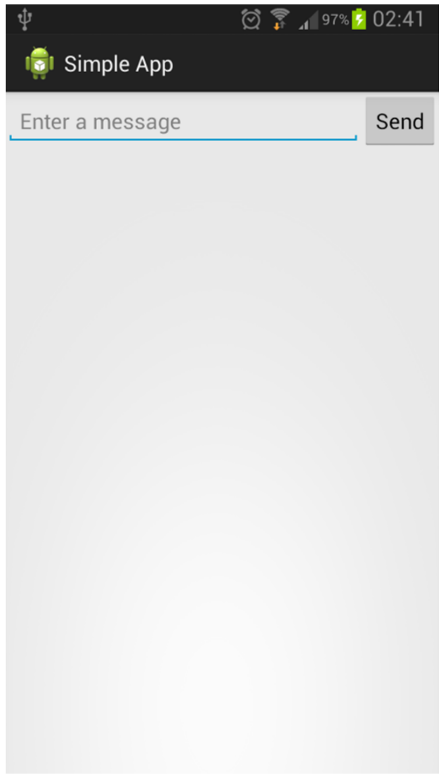
- The “main” Activity.
- “New” Activity is pushed onto the back stack and takes user focus.
- When the user is done with the current activity and presses the Back button, it is popped from the stack (and destroyed) and the previous activity resumes.



Android Course

Back Stack

- The “main” Activity.
- “New” Activity is pushed onto
  the back stack and takes user focus.
- When the user is done with the current
  activity and presses the Back button,
  it is popped from the stack (and destroyed)
  and the previous activity resumes.



Activity Lifecycle

- When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback methods.
- There are several callback methods that an activity might receive, due to a change in its state—whether the system is creating it, stopping it, resuming it, or destroying it.
- Each callback provides you the opportunity to perform specific work that's appropriate to that state change.

Activity Lifecycle States

- **Created:** Transient and the system quickly moves to the next state.
- **Started:** Transient and the system quickly moves to the next state.
- **Resumed** (Running)**:** The activity is in the foreground and the user can interact with it.
- **Paused:** The activity is partially obscured by another activity (semi-transparent or doesn't cover the entire screen). The paused activity does not receive user input and cannot execute any code.
- **Stopped:** The activity is completely hidden and not visible to the user; it is considered to be in the background. While stopped, the activity instance and all its state information such as member variables is retained, but it cannot execute any code.

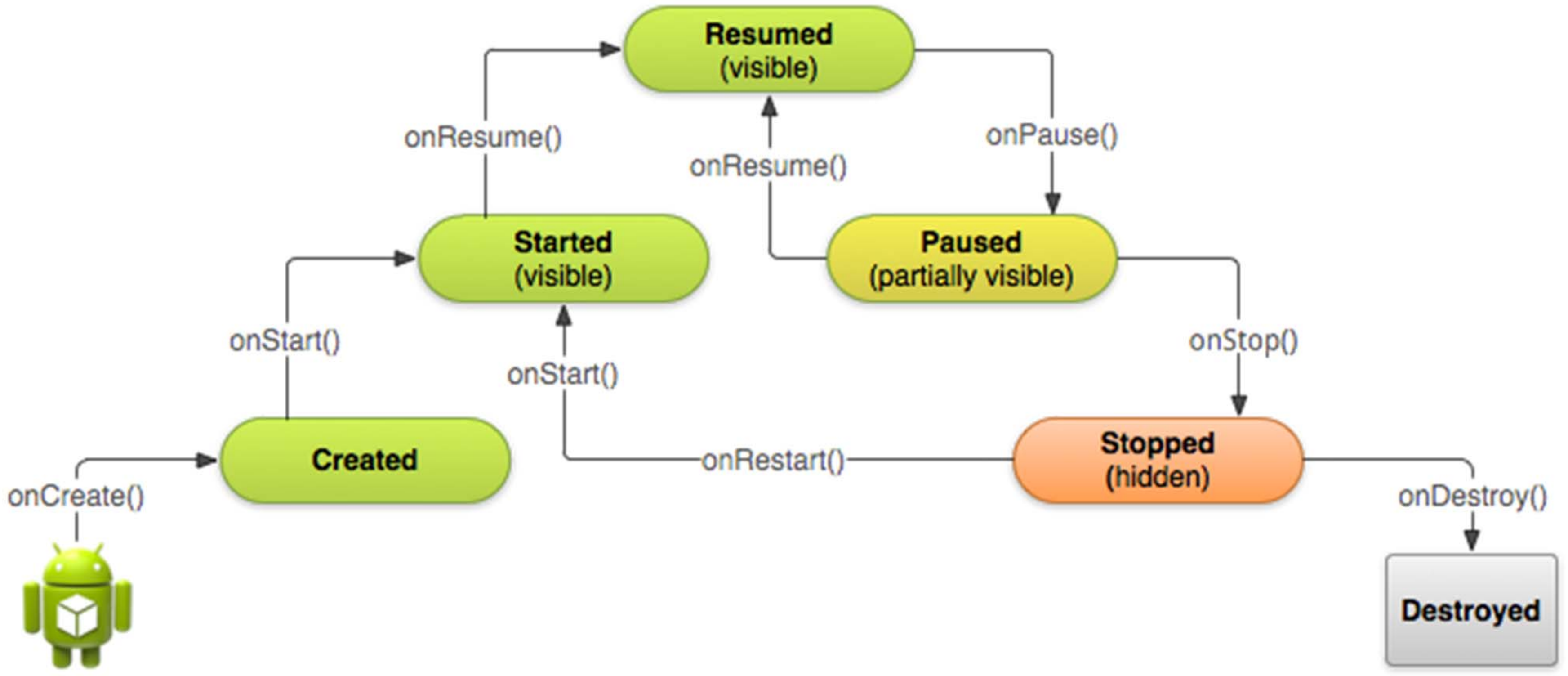
Activity Lifecycle States

There are several situations in which an activity transitions between different states:

- However, only three of these states can be static. That is, the activity can exist in one of only three states for an extended period of time: Resumed, Paused, and Stopped

- The other states (Created and Started) are transient and the system quickly moves from them to the next state by calling the next lifecycle callback method.

Activity Lifecycle States & Callbacks



Callback Methods

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

Callback Methods: onCreate()

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

- Called when the activity is first created.
- This is where you should do all of your normal static set up — create views, bind data to lists, and so on.
- Always followed by onStart().

Callback Methods: OnRestart()

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

- Called after the activity has been stopped, just prior to it being started again.
- Always followed by onStart()

Callback Methods: OnStart()

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

- Called just before the activity becomes visible to the user.
- Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.

Callback Methods: OnResume()

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

- Called just before the activity starts interacting with the user.
- At this point the activity is at the top of the activity stack, with user input going to it.
- Always followed by onPause().

Callback Methods: OnPause()

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

- Called when the system is about to start resuming another activity.
- This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on.
- It should do whatever it does very quickly, because the next activity will not be resumed until it returns.
- Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.

Callback Methods: OnStop()

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

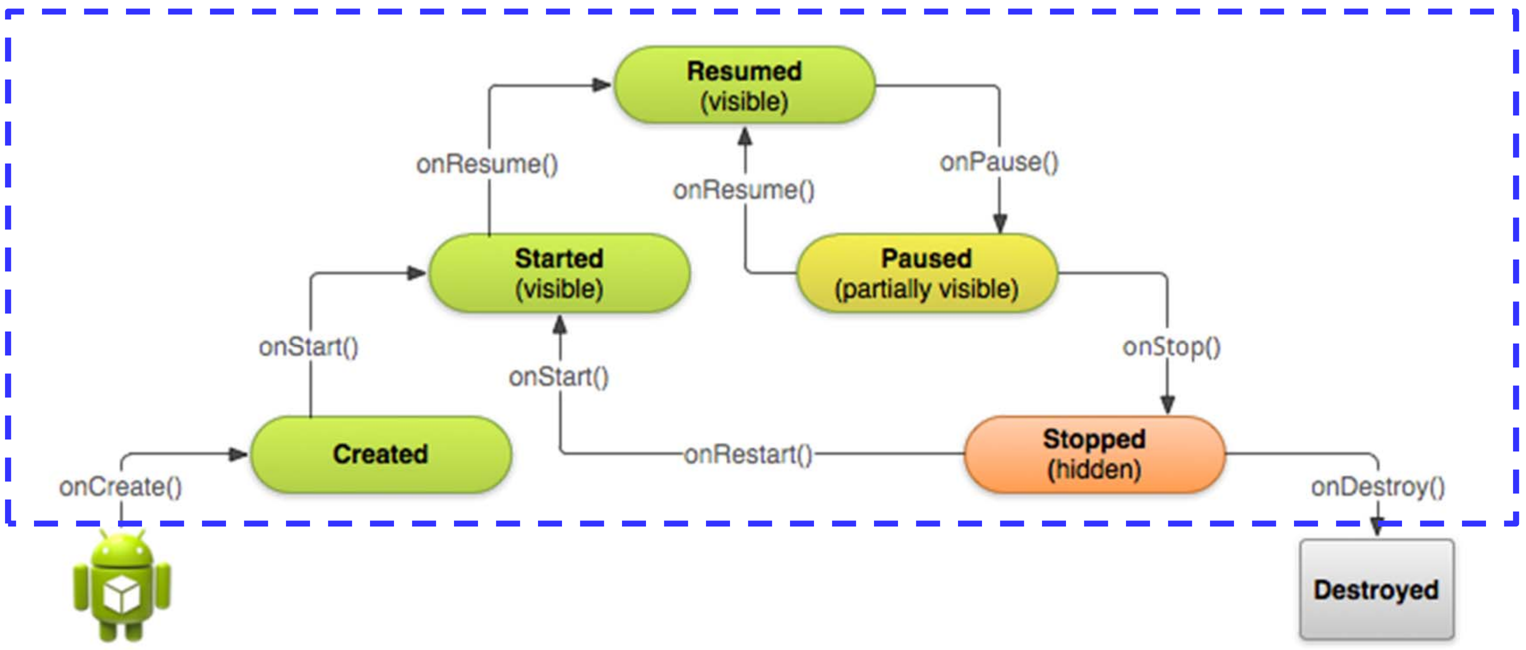
- Called when the activity is no longer visible to the user.
- This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.
- Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.

Callback Methods: OnDestroy()

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

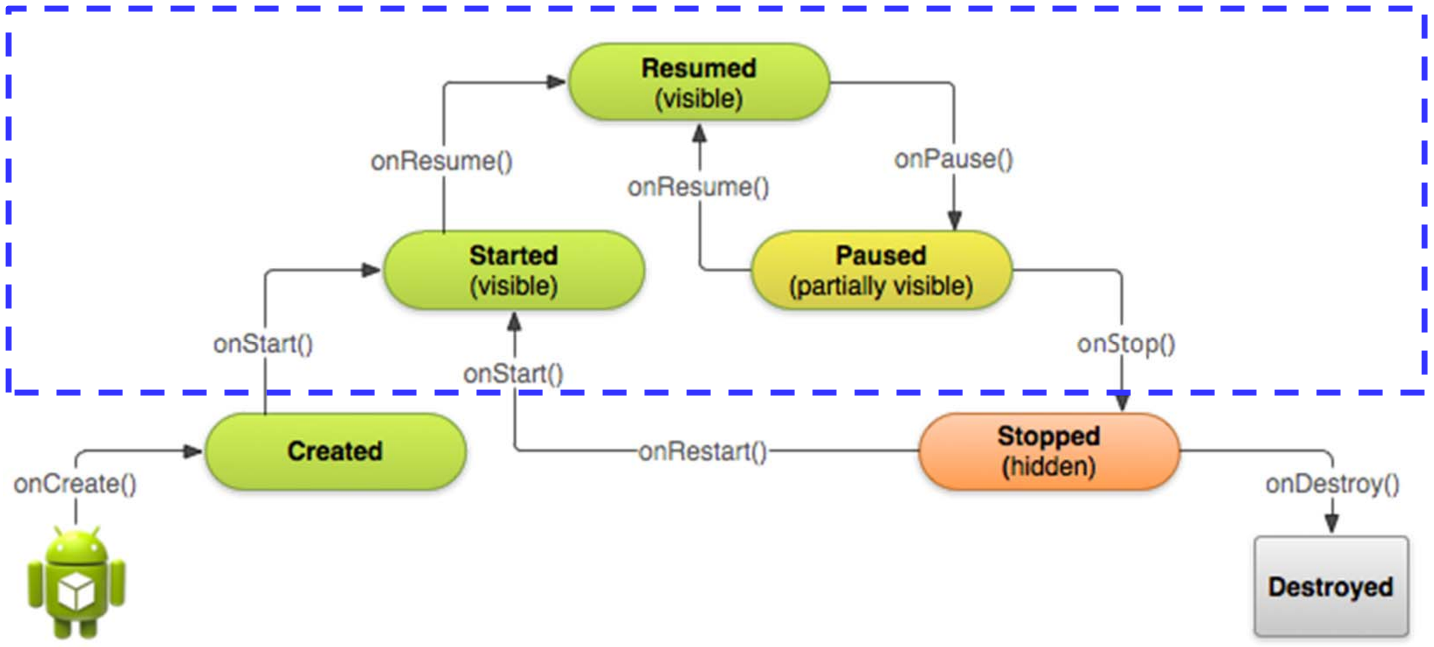
- Called before the activity is destroyed.
- This is the final call that the activity will receive.
- It could be called either because the activity is finishing (someone called finish() on it), or because the system is temporarily destroying this instance of the activity to save space.
- You can distinguish between these two scenarios with the isFinishing() method.

The Entire Lifetime



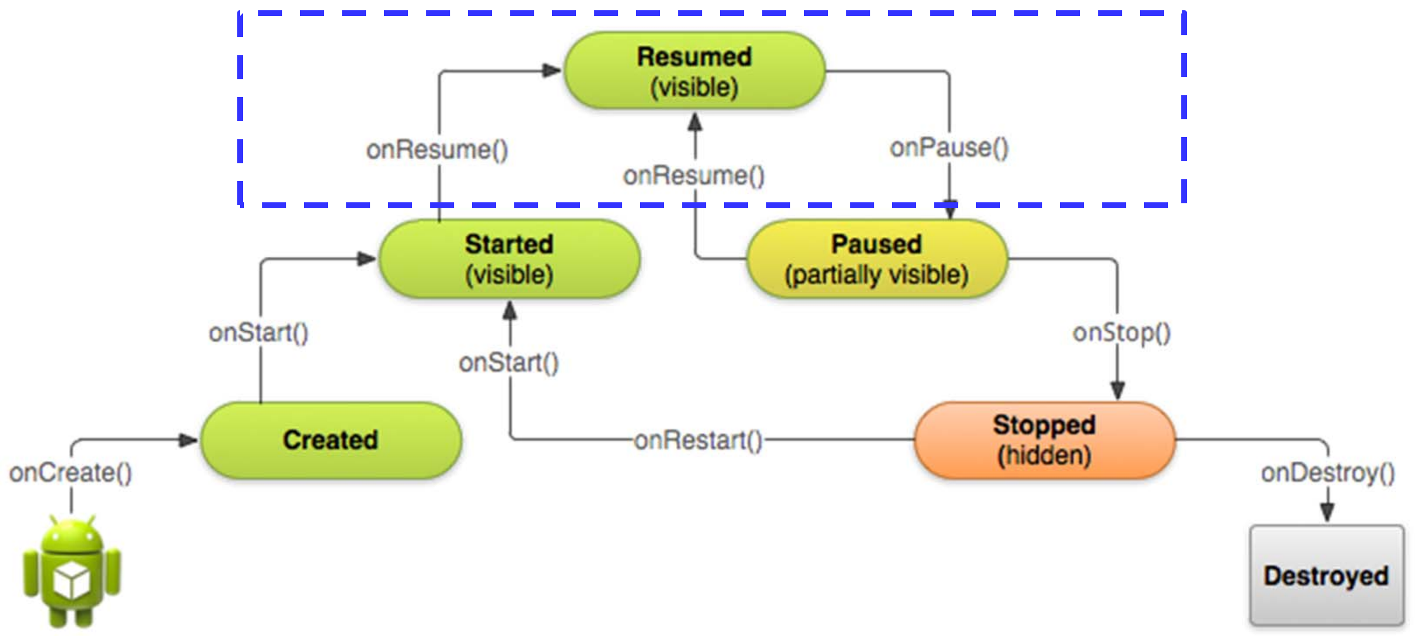
The entire lifetime of an activity happens between the call to onCreate() and the call to onDestroy(). Your activity should perform setup of "global" state (e.g., layout) in onCreate(), and release all remaining resources in onDestroy().

The Visible Lifetime



The visible lifetime of an activity happens between the call to onStart() and the call to onStop(). During this time, the user can see the activity on-screen and interact with it.

The Foreground Lifetime



The foreground lifetime of an activity happens between the call to onResume() and the call to onPause(). During this time, the activity is in front of all other activities on screen and has user input focus.

Implementing Callback Methods

- All of the callback methods are hooks that **you can override** to do appropriate work when the state of your activity changes.
- Your implementation of these lifecycle methods **must always call the superclass implementation** before doing any work.

Implementing Callback Methods

```
public class ExampleActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
        . . .
   }

   . . .
}
```

Implementing Callback Methods

```
public class ExampleActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
        . . .
   }

   @Override
   public void onStart() {
        super.onStart();
        // The activity is about to become visible.
        . . .
   }
   . . .
}
```

Implementing Callback Methods

```
public class ExampleActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
        . . .
   }

   @Override
   public void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
        . . .
   }
   . . .
}
```

Implementing Callback Methods

```
public class ExampleActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
        . . .
   }

   @Override
   public void onPause() {
        super.onPause();
        // Another activity is taking focus.
        . . .
   }
   . . .
}
```

Implementing Callback Methods

```
public class ExampleActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
        . . .
   }

   @Override
   public void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
        . . .
   }
   . . .
}
```

Implementing Callback Methods

```
public class ExampleActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
        . . .
   }

   @Override
   public void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
        . . .
   }
   . . .
}
```

Implementing Callback Methods

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

Using Log Utility to Test Callback

- You can use Log utility to test when Callback Methods are called during Activity's Lifecycle.
- `Log.d(“TAG",“Some Message");`

USING INTENTS

Intent

- An Intent is a messaging object you can use to request an action from another app component.

- There are two types of intents:

  - Implicit intents

  - Explicit intents

Implicit Intent

- Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

- For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Implicit Intent

```
String url="http://google.com";
Intent intent=new Intent(Intent.ACTION_VIEW,Uri.parse(url));
startActivity(intent);
```

Explicit Intent

- Explicit intents specify the component to start by name (the fully-qualified class name).
- You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.

Explicit Intent

```
Intent intent=new Intent(this, SecondActivity.class);
startActivity(intent);
```

Passing Extra Key-Value Information

```
Intent intent=new Intent(this, SecondActivity.class);
intent.putExtra("v1", "Some Value");
intent.putExtra("v2", "Another Value");
startActivity(intent);
```

Passing Extra Key-Value Information

```
public class SecondActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent intent=getIntent();
        String msg1=intent.getStringExtra("v1");
        String msg2=intent.getStringExtra("v2");

        . . .
   }
. . .
}
```

Start Activity For Result

- Starting another activity doesn't have to be one-way. You can also start another activity and receive a result back. To receive a result, call startActivityForResult() (instead of startActivity()).

- There's nothing special about the Intent object you use when starting an activity for a result, but you do need to pass an additional integer argument to the startActivityForResult() method.

- The integer argument is a "request code" that identifies your request. When you receive the result Intent, the callback provides the same request code so that your app can properly identify the result and determine how to handle it.

Start Second Activity For Result

```
Intent intent=new Intent(this, SecondActivity.class);
startActivityForResult(intent, 2);
```

Send Results From Second Activity

```
. . .
    String message=editText1.getText().toString();

    Intent intent=new Intent();
    intent.putExtra("MESSAGE",message);

    setResult(2,intent);

    finish(); //finishing activity
. . .
```

Get Result in First Activity

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
   Intent data)
{
  super.onActivityResult(requestCode, resultCode, data);

  // check if the request code is same as what is passed
  if(requestCode==2)
  {
    String message=data.getStringExtra("MESSAGE");
    . . .

  }

}
```

EXAMPLES:
IMPLICIT INTENTS

Implicit Intent (View Web Page)

```
String url="http://google.com";

Intent browserIntent=new
   Intent(Intent.ACTION_VIEW,Uri.parse(url));

startActivity(browserIntent);
```

Implicit Intent (Dial/Call Phone)

```
String strPhone="0518446000";

Intent callIntent = new Intent(Intent.ACTION_DIAL);

// Intent callIntent = new Intent(Intent.ACTION_CALL);
// You will also need to set permissions in AndroidMenifest.xml File
// <uses-permission android:name="android.permission.CALL_PHONE" />

callIntent.setData(Uri.parse("tel:" + strPhone));
startActivity(callIntent);
```

Implicit Intent (SMS)

```
String strPhone="0518446000";
String strMessage="Some Message ...";

Uri smsUri = Uri.parse("tel:" + strPhone);
Intent smsIntent = new Intent(Intent.ACTION_VIEW, smsUri);

smsIntent.putExtra("address", strPhone);
smsIntent.putExtra("sms_body", strMessage);
smsIntent.setType("vnd.android-dir/mms-sms");

startActivity(smsIntent);
```

Implicit Intent (Email)

```
String strEmail="someone@somewhere.com";
String strMessage="Some Message ...";
String strSubject="Some Subject";

Intent emailIntent = new Intent(Intent.ACTION_SEND);

emailIntent.setType("plain/text");
emailIntent.putExtra(Intent.EXTRA_EMAIL, new String[]{strEmail});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, strSubject);
emailIntent.putExtra(Intent.EXTRA_TEXT, strMessage);

startActivity(Intent.createChooser(emailIntent, ""));
```

References

- http://developer.android.com/guide/components/activities.html
- http://developer.android.com/training/basics/activity-lifecycle/index.html
- http://developer.android.com/guide/components/intents-filters.html
- http://developer.android.com/training/basics/activity-lifecycle/recreating.html
- https://developer.android.com/guide/components/intents-common.html

Q & A