



Lesson 3

Application's Life Cycle

Victor Matos
Cleveland State University

Anatomy of Android Applications

An Android application consists of one or more *core components*.

Like musicians in an orchestra, the independent core components cooperate with each other contributing somehow to the success of the application.

A core component can be:

- 1. *An Activity***
- 2. *A Service***
- 3. *A broadcast receiver***
- 4. *A content provider***



Anatomy of Android Applications

1. Activity

- A typical Android **application** consists of *one or more activities*.
- An activity is roughly equivalent to a Windows-Form .
- An *activity* usually shows a *single visual user interface* (UI).
- Only one activity is chosen to be executed first when the application is launched.
- An activity may transfer control and data to another activity through an interprocess communication protocol called **intents**.

Anatomy of Android Applications

2. Service

- Services are a special type of activity that *do not have a visual user interface*.
- Services usually run in the background for an indefinite period of time.
- Applications start their own services or connect to services already active.
- **For example**, your background GPS service could be set to inconspicuously run in the background detecting satellite, phone tower or wi-fi router location information. The service periodically broadcast location coordinates to any application listening for that kind of data. An application may bind to the running GPS service or be the first to execute it.

Anatomy of Android Applications

3. Broadcast receiver

- A **BroadcastReceiver** is a dedicated listener that waits for system-wide or locally transmitted messages.
- *Broadcast receivers do not display a user interface.*
- They typically register with the system by means of a filter acting as a key. When the broadcasted message matches the key the receiver is activated.
- A broadcast receiver could respond by either executing a specific activity or use the *notification* mechanism to request the user's attention.

Anatomy of Android Applications

4. Content provider

- A *content provider* is a data-centric service that makes persistent datasets available to any number of applications.
- Common global datasets include: contacts, pictures, messages, audio files, emails.
- The global datasets are usually stored in a SQLite database.
- The content provider class offers a standard set of “database-like” methods to enable other applications to retrieve, delete, update, and insert data items.

Application's Life Cycle

Each Android application runs inside its own instance of a Dalvik Virtual Machine (DVM).

At any point in time several parallel DVM instances could be active.

Unlike a common Windows or Unix process, an Android application does not *completely* controls the completion of its lifecycle.

Occasionally hardware resources may become critically low and the OS could order early termination of any process. The decision considers factors such as:

1. Number and age of the application's components currently running,
2. relative importance of those components to the user, and
3. how much free memory is available in the system.


Component Lifecycles

All components execute according to a master plan that consists of:

1. A **beginning** - responding to a request to instantiate them
2. An **end** - when the instances are destroyed.
3. A sequence of **in between** states – components sometimes are *active* or *inactive*, or in the case of activities - *visible* or *invisible*.



Activity Stack

- Activities in the system are scheduled using an ***activity stack***.
- When a new activity is *started*, it is placed on *top* of the stack to become the *running* activity
- The previous activity is pushed-down one level in the stack, and may come back to the foreground once the new activity finishes.
- If the user presses the *Back Button*  the current activity is terminated and the next activity on the stack moves up to become active.

Activity Stack

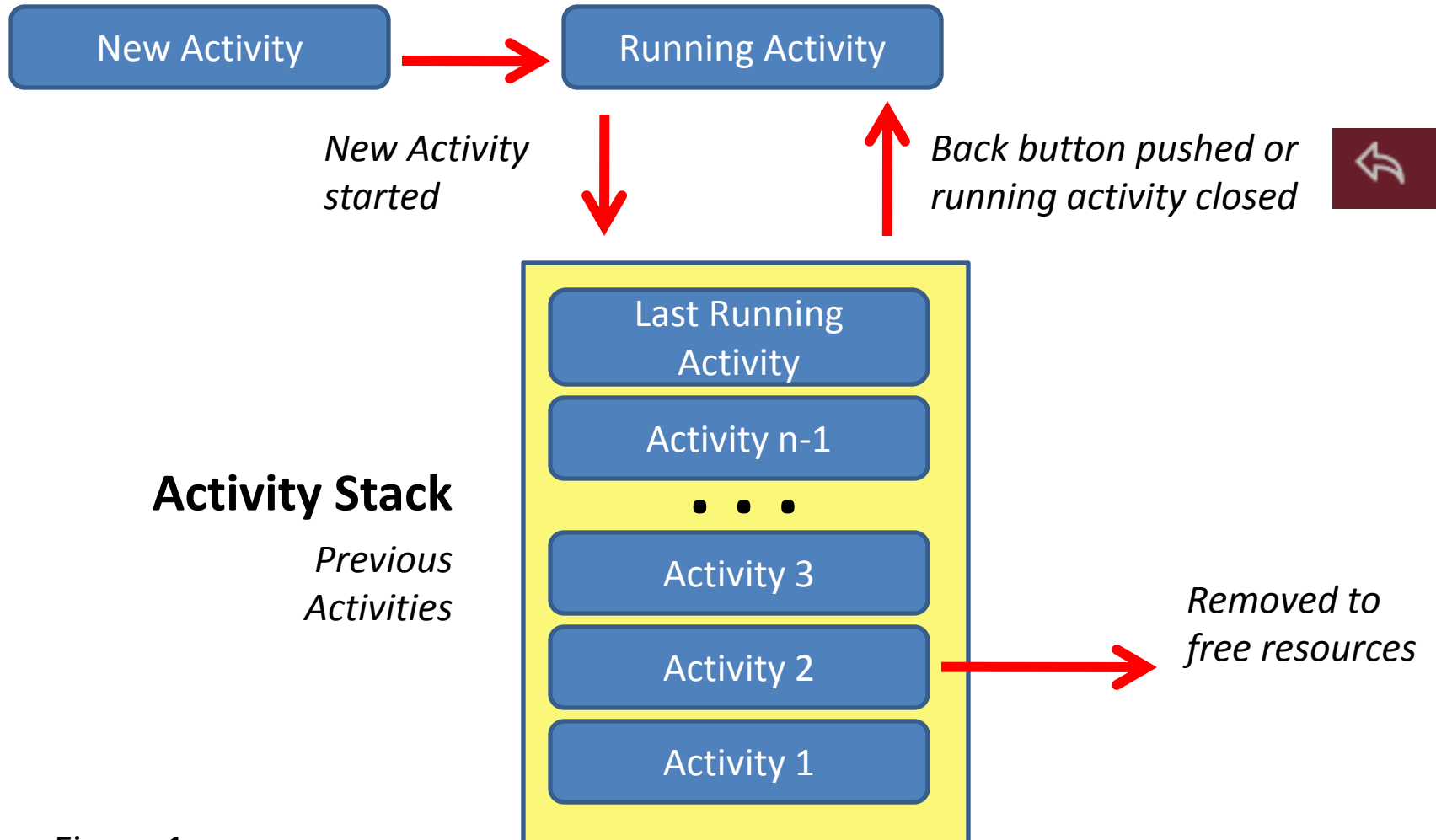


Figure 1.

Life Cycle Events

Life Cycle States

When progressing from one state to the other, the OS notifies the application of the changes by issuing calls to the following protected *transition methods*:

```
void onCreate(Bundle savedInstanceState)  
void onStart()  
void onRestart()  
void onResume()
```

```
void onPause()  
void onStop()  
void onDestroy()
```

Life Cycle Callbacks

Most of your code
goes here

```
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").
```

```
}
```

Save your
important data
here

```
@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.
```

```
}
```

```
}
```

Reference:

<http://developer.android.com/reference/android/app/Activity.html>

Life Cycle States

An activity has essentially three states:

1. It is *active or running*
2. It is *paused* or
3. It is *stopped* .

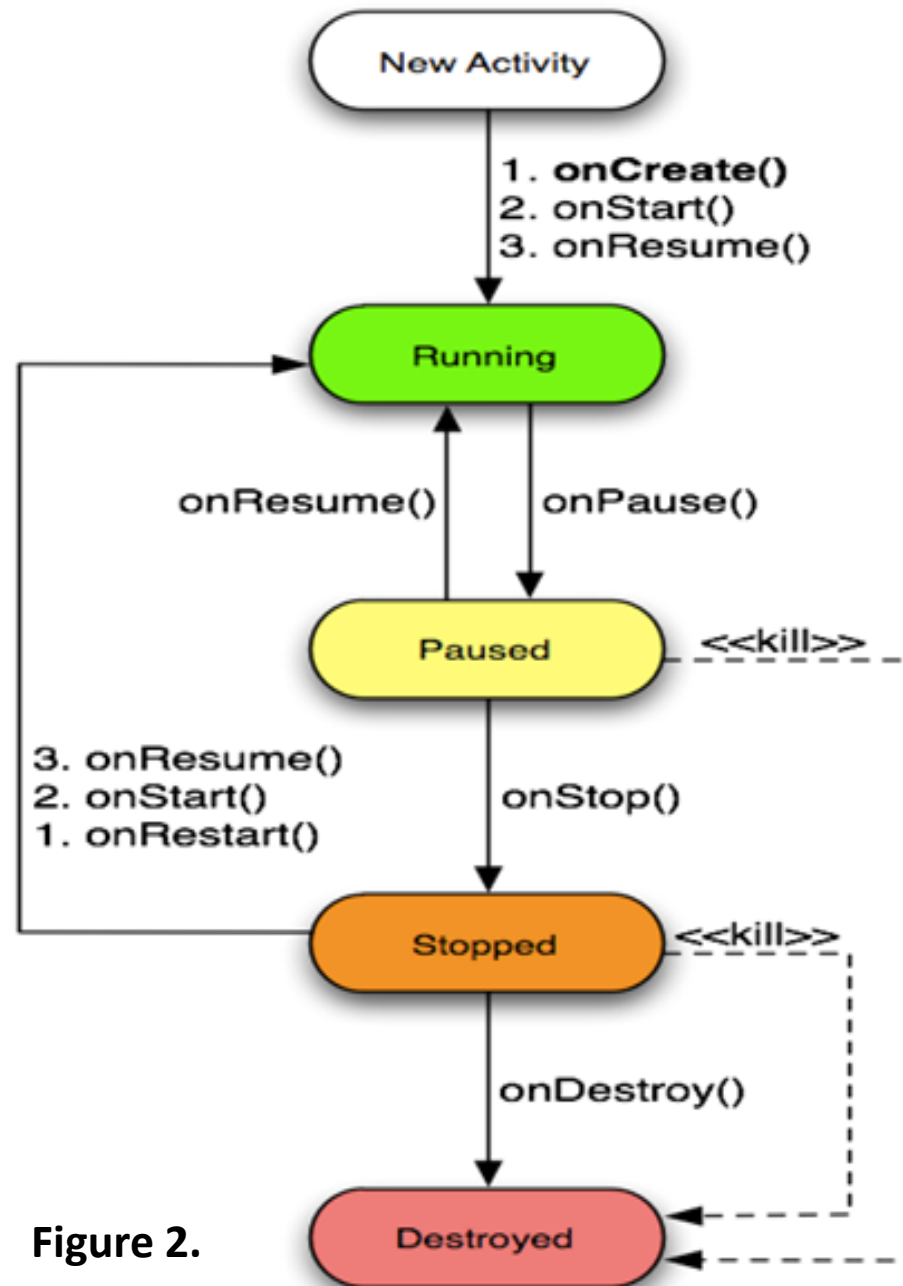


Figure 2.

Life Cycle States

An activity has essentially three states:

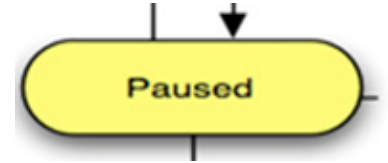


1. It is **active or running** when it is in the *foreground* of the screen (at the top of the *activity stack*).

This is the activity that has “focus” and its graphical interface is responsive to the user’s interactions.

Life Cycle States

An activity has essentially three states (cont.) :



2. It is **paused** if it has lost focus but is still visible to the user.

That is, another activity lies on top of it and that new activity either is *transparent* or *doesn't cover the full screen*.

A paused activity is *alive* (maintaining its state information and attachment to the window manager).

Paused activities can be killed by the system when available memory becomes extremely low.

Life Cycle States

An activity has essentially three states (cont.):



3. It is **stopped** if it is completely *obscured* by another activity.

Continues to retain all its state information.

It is no longer visible to the user (its window is hidden and its life cycle could be terminated at any point by the system if the resources that it holds are needed elsewhere).

Application's Life Cycle

Your turn!

EXPERIMENT 1.



Teaching notes

1. Write an Android app to show the different cycles followed by an application.
2. The **main.xml** layout should include a Button (text: "Finish", id: btnFinish) and an EditText container (txt: "" and id txtMsg).
3. Use the onCreate method to connect the button and textbox to the program.
Add the following line of code:

```
Toast.makeText (this, "onCreate", 1).show();
```
4. The click method has only one command: **finish()**; called to terminate the application. Add a Toast-command (as the one above) to each of the remaining six main events. To simplify your job use the Eclipse's top menu: Source > Override/Implement Methods...
5. On the option window check mark each of the following events: onStart, onResume, onPause, onStop, onDestroy, onRestart (notice how many *onEvent...* methods are there!!!)
6. Save your code.



Application's Life Cycle

Your turn!

EXPERIMENT 1 (cont.)



Teaching notes

7. Compile and execute application.
8. Write down the sequence of messages displayed by the Toast-commands.
9. Press the FINISH button. Observe the sequence of states.
10. Re-execute the application
11. Press emulator's HOME button. What happens?
12. Click on launch pad, look for icon and return to the app. What sequence of messages is displayed?
13. Click on the emulator's CALL (Green phone). Is the app paused or stopped?
14. Click on the BACK button to return to the application.
15. Long-tap on the emulator's HANG-UP button. What happens?

Application's Life Cycle

Your turn!

EXPERIMENT 2



Teaching notes

7. Run a second emulator.
 1. Make a voice-call to the first emulator that is still showing our app. What happens on this case? (real-time synchronous request)
 2. Send a text-message to first emulator (asynchronous attention request)
8. Write a phrase in the EditText box (“these are the best moments of my life...”).
9. Re-execute the app. What happened to the text?

Application's Life Cycle

Your turn!

EXPERIMENT 3



Teaching notes

Provide data persistency.

18. Use the **onPause** method to add the following fragment

```
SharedPreferences myFile1 = getSharedPreferences("myFile1",
                                                Activity.MODE_PRIVATE);

SharedPreferences.Editor myEditor = myFile1.edit();
String temp = txtMsg.getText().toString();
myEditor.putString("mydata", temp);
myEditor.commit();
```

18. Use the **onResume** method to add the following fragment

```
SharedPreferences myFile = getSharedPreferences("myFile1",
                                                Activity.MODE_PRIVATE);

if ( (myFile != null) && (myFile.contains("mydata")) ) {
    String temp = myFile.getString("mydata", "***");
    txtMsg.setText(temp);
}
```

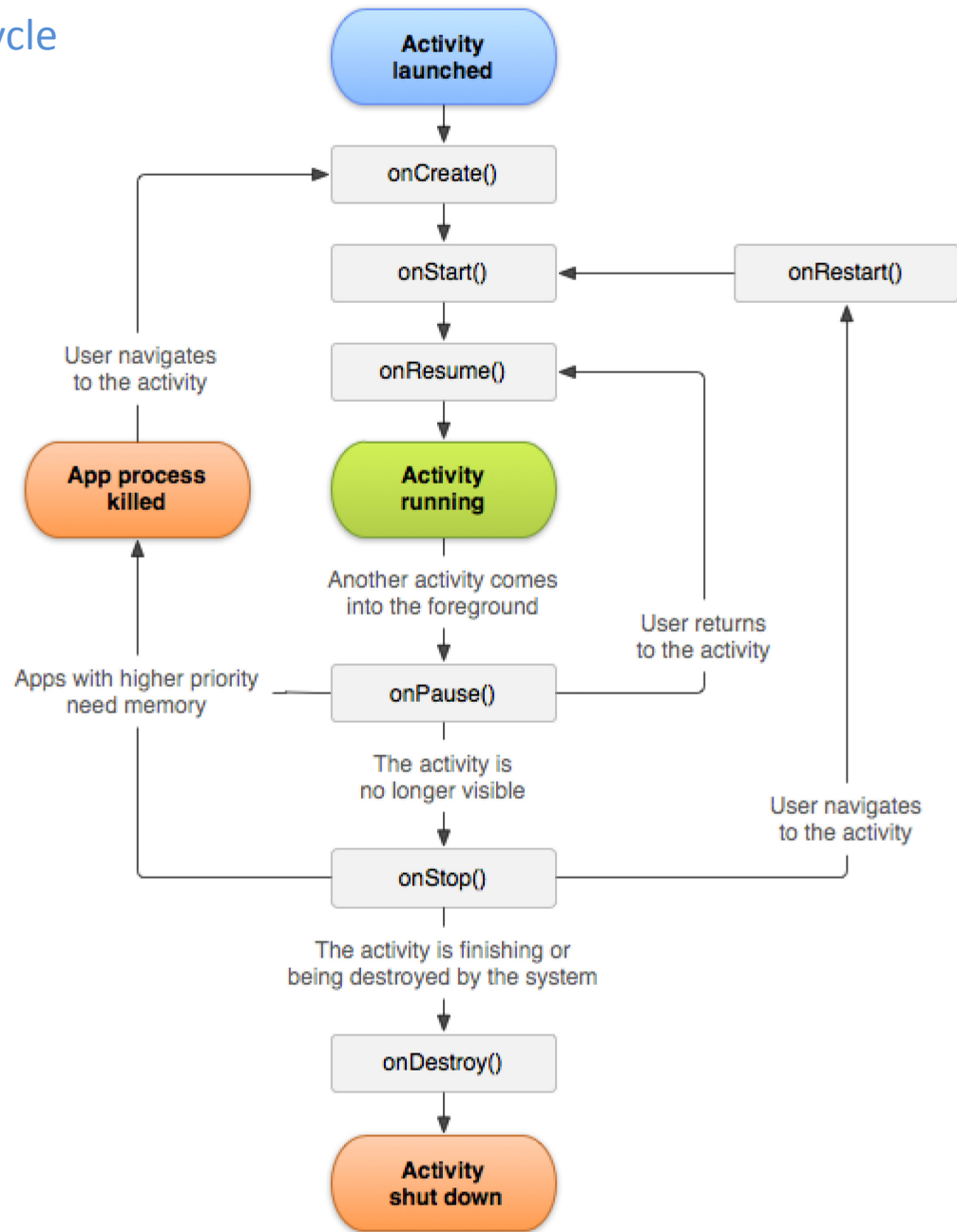
19. What happens now with the data previously entered in the text box?



3. Android – Application's Life Cycle

Application's Life Cycle

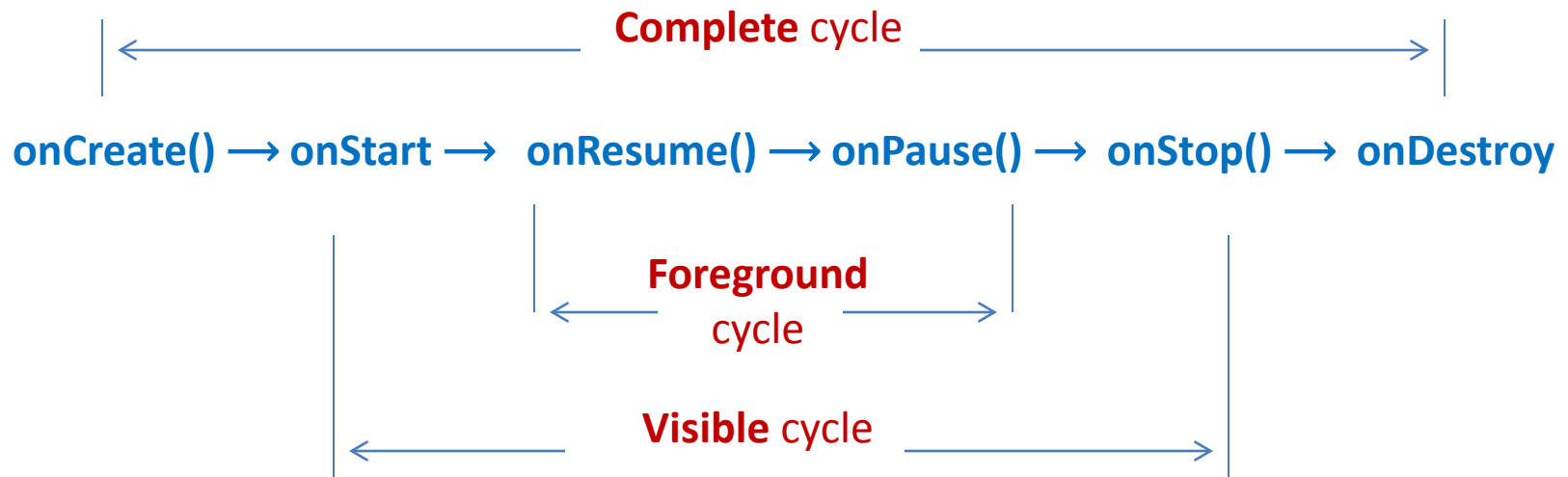
Figure 3.



Application's Lifetime

Complete / Visible / Foreground Lifetime

- An activity begins its lifecycle when entering the **onCreate()** state .
- If not interrupted or dismissed, the activity performs its job and finally terminates and releases its acquired resources when reaching the **onDestroy()** event.



Life Cycle Events

Associating Lifecycle Events with Application's Code

Applications do not need to implement each of the transition methods, however there are mandatory and recommended states to consider

(Mandatory)

All activities must implement **onCreate()** to do the initial setup when the object is first instantiated.

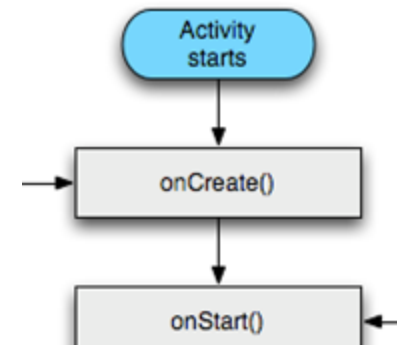
(Highly Recommended)

Activities should implement **onPause()** to commit data changes in anticipation to stop interacting with the user.

Life Cycle Methods

Method: **onCreate()**

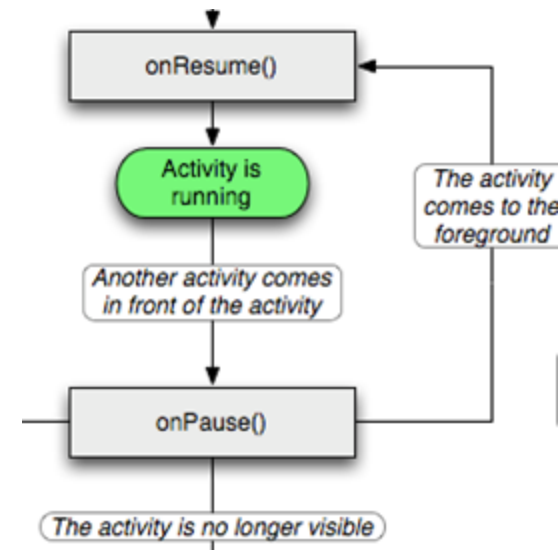
- Called when the activity is first created.
- Most of your application's code is written here.
- Typically used to define listener's behavior, initialize data structures, wire-up UI view elements (buttons, text boxes, lists) with static Java controls, etc.
- It may receive a data *Bundle* object containing the activity's previous state (if any).
- Followed by *onStart()*



Life Cycle Methods

Method: **onPause()**

1. Called when the system is about to transfer control to another activity.
2. Gives you a chance to *commit* unsaved data, and stop work that may unnecessarily burden the system.
3. The next activity waits until completion of this state.
4. Followed either by *onResume()* if the activity returns back to the foreground, or by *onStop()* if it becomes invisible to the user.
5. A paused activity could be *killed* by the system.



Life Cycle Methods

Killable States

- Activities on killable states can be terminated by the system when memory resources become critically low.
- Methods: `onPause()`, `onStop()`, and `onDestroy()` are *killable*.
- `onPause()` is the only state that is *guaranteed* to be given a chance to complete before the process is killed.
- You should use `onPause()` to write any pending persistent data.

Life Cycle Methods

As an aside...

Android Preferences

Preferences is a simple Android *persistence* mechanism used to store and retrieve *key-value* pairs of primitive data types. Similar to a Java Hasmap. Suitable for keeping small amounts of state data.

```
SharedPreferences myPrefSettings =  
    getSharedPreferences(MyPreferenceFile, actMode);
```

- A named *preferences file* could be shared with other components in the same application.
- An anonymous `Activity.getSharedPreferences()` is used only by the calling activity.
- You cannot share preferences across applications.

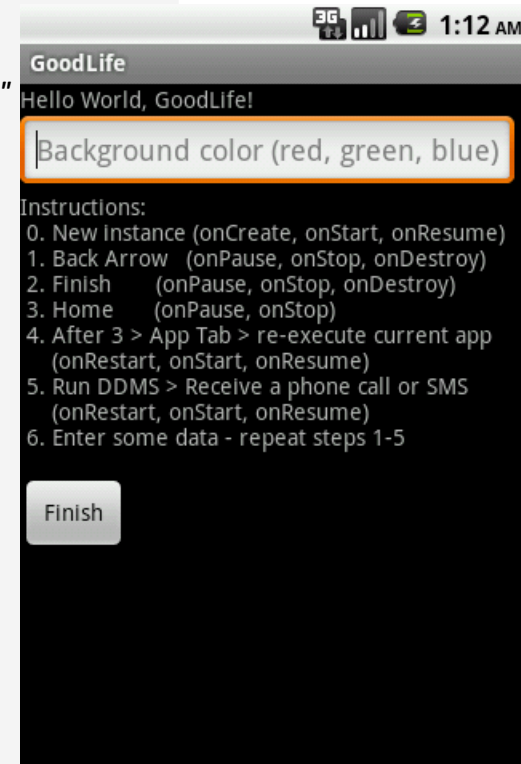
Example Life Cycle

Example

The following application demonstrates some of the state transitioning situations experienced in the life-cycle of a typical Android activity.

LAYOUT

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myScreen"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff000000"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
    <EditText
        android:id="@+id/txtColorSelect"
        android:hint="Background color (red, green, blue)"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </EditText>
    <TextView
        android:id="@+id/txtToDo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#00000000">
    <!-- transparent -->
    </TextView>
    <Button
        android:text=" Finish "
        android:id="@+id/btnFinish"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </Button>
</LinearLayout>
```



Example: Life Cycle

Code: Life Cycle Demo. Part 1

```
Package cis493.lifecycle
```

```
import android.app.Activity;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.*;
```

```
//GOAL: show the following life-cycle events in action
```

```
//protected void onCreate(Bundle savedInstanceState) ;  
//protected void onStart() ;  
//protected void onRestart() ;  
//protected void onResume() ;  
//protected void onPause() ;  
//protected void onStop() ;  
//protected void onDestroy() ;
```

Example: Life Cycle

Code: Life Cycle Demo. Part 1

```
public class GoodLife extends Activity {  
  
    // class variables and constants  
    public static final String MYPREFSID = "MyPrefs001";  
    public static final int actMode = Activity.MODE_PRIVATE;  
  
    LinearLayout  myScreen;  
    EditText  txtColorSelect;  
    TextView  txtToDo;  
    Button  btnFinish;
```

Example: Life Cycle

Code: Life Cycle Demo. Part 2

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    myScreen = (LinearLayout) findViewById(R.id.myScreen);

    txtToDo = (TextView) findViewById(R.id.txtToDo);
    String msg = "Instructions:                                \n "
        + "0. New instance (onCreate, onStart, onResume)      \n "
        + "1. Back Arrow    (onPause, onStop, onDestroy)        \n "
        + "2. Finish          (onPause, onStop, onDestroy)          \n "
        + "3. Home (onPause, onStop)                                  \n "
        + "4. After 3 > App Tab > re-execute current app            \n "
        + "    (onRestart, onStart, onResume)                        \n "
        + "5. Run DDMS > Receive a phone call or SMS                 \n "
        + "    (onRestart, onStart, onResume)                        \n "
        + "6. Enter some data - repeat steps 1-5                     \n ";

    txtToDo.setText(msg);
```

Example: Life Cycle

Code: Life Cycle Demo. Part 2

```
txtColorSelect = (EditText) findViewById(R.id.txtColorSelect);
// you may want to skip discussing the listener until later
→ txtColorSelect.addTextChangedListener(new TextWatcher() {
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        // TODO Auto-generated method stub
    }
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
        // TODO Auto-generated method stub
    }
    public void afterTextChanged(Editable s) {
        → changeBackgroundColor(s.toString());
    }
});

btnFinish = (Button) findViewById(R.id.btnFinish);
btnFinish.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
        finish();
    }
});
Toast.makeText(getApplicationContext(), "onCreate", 1).show();
}
```


Example: Life Cycle

Code: Life Cycle Demo. Part 3


```
@Override
protected void onPause() {
    super.onPause();
    saveDataFromCurrentState();
    Toast.makeText(this, "onPause", 1).show();
}

@Override
protected void onRestart() {
    super.onRestart();
    Toast.makeText(this, "onRestart", 1).show();
}

@Override
protected void onResume() {
    super.onResume();
    Toast.makeText(this, "onResume", 1).show();
}
```

Example: Life Cycle

Code: Life Cycle Demo. Part 4



```
@Override
protected void onStart() {
    // TODO Auto-generated method stub
    super.onStart();
    updateFromSavedState();
    Toast.makeText(this, "onStart", 1).show();
}

@Override
protected void onDestroy() {
    // TODO Auto-generated method stub
    super.onDestroy();
    Toast.makeText(this, "onDestroy", 1).show();
}

@Override
protected void onStop() {
    // TODO Auto-generated method stub
    super.onStop();
    Toast.makeText(this, "onStop", 1).show();
}
```

Example: Life Cycle

Code: Life Cycle Demo. Part 5

```
protected void saveDataFromCurrentState() {
    SharedPreferences myPrefs = getSharedPreferences(MYPREFSID, actMode);
    SharedPreferences.Editor myEditor = myPrefs.edit();
    myEditor.putString("myBkColor", txtColorSelect.getText().toString());
    myEditor.commit();
} // saveDataFromCurrentState

protected void updateFromSavedState() {
    SharedPreferences myPrefs = getSharedPreferences(MYPREFSID, actMode);

    if ((myPrefs != null) && (myPrefs.contains("myBkColor"))) {
        String theChosenColor = myPrefs.getString("myBkColor", "");
        txtColorSelect.setText(theChosenColor);
        changeBackgroundColor(theChosenColor);
    }
} // UpdateFromSavedState

protected void clearMyPreferences() {
    SharedPreferences myPrefs = getSharedPreferences(MYPREFSID, actMode);
    SharedPreferences.Editor myEditor = myPrefs.edit();
    myEditor.clear();
    myEditor.commit();
}
```

Example: Life Cycle

Code: Life Cycle Demo. Part 6

```
private void changeBackgroundColor (String theChosenColor){  
    // change background color  
    if (theChosenColor.contains("red"))  
        myScreen.setBackgroundColor(0xffff0000) ;  
    else if (theChosenColor.contains("green"))  
        myScreen.setBackgroundColor(0xff00ff00) ;  
    else if (theChosenColor.contains("blue"))  
        myScreen.setBackgroundColor(0xff0000ff) ;  
    else {  
        //reseting user preferences  
        clearMyPreferences() ;  
        myScreen.setBackgroundColor(0xff000000) ;  
    }  
}
```

Example: Life Cycle

Code: Life Cycle Demo. Part 8

```
/*
protected void onRestoreInstanceState(Bundle savedInstanceState)
This method is called after onStart() when the activity is being re-initialized
from a previously saved state.
The default implementation of this method performs a restore of any view state
that had previously been frozen by onSaveInstanceState(Bundle).
*/
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    Toast.makeText(getApplicationContext(),
        "onRestoreInstanceState ...BUNDLING",
        Toast.LENGTH_LONG).show();
}
```

Example: Life Cycle

Code: Life Cycle Demo. Part 9

```
/*  
protected void onSaveInstanceState(Bundle outState)
```

Called to retrieve per-instance state from an activity before being killed so that the state can be restored in

*onCreate(Bundle) or
onRestoreInstanceState(Bundle) (the Bundle populated by this method will be passed to both).*

*This method is called before an activity may be killed so that when it comes back some time in the future it can restore its state. For example, if activity B is launched in front of activity A, and at some point activity A is killed to reclaim resources, activity A will have a chance to save the current state of its user interface via this method so that when the user returns to activity A, the state of the user interface can be restored via:
onCreate(Bundle) or onRestoreInstanceState(Bundle).*

```
*/
```

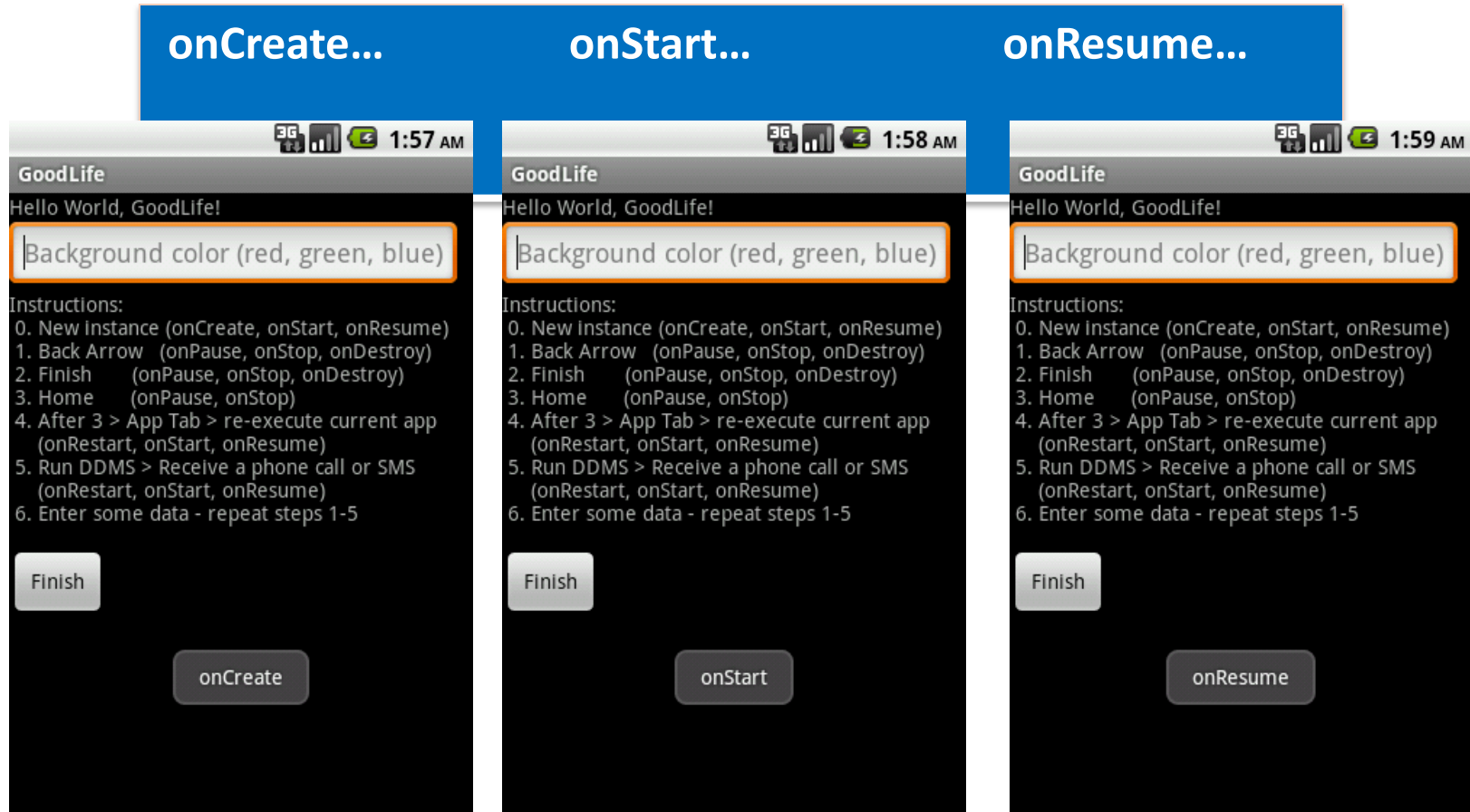
Example: Life Cycle

Code: Life Cycle Demo. Part 10

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    Toast.makeText(getApplicationContext(),
        "onSaveInstanceState ...BUNDLING",
        Toast.LENGTH_LONG).show();
} // onSaveInstanceState

} // LyfeCicleDemo
```

Example: Life Cycle

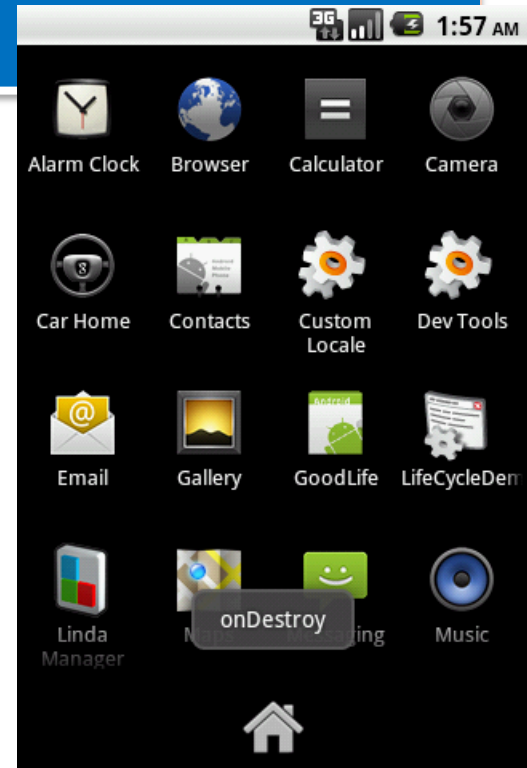
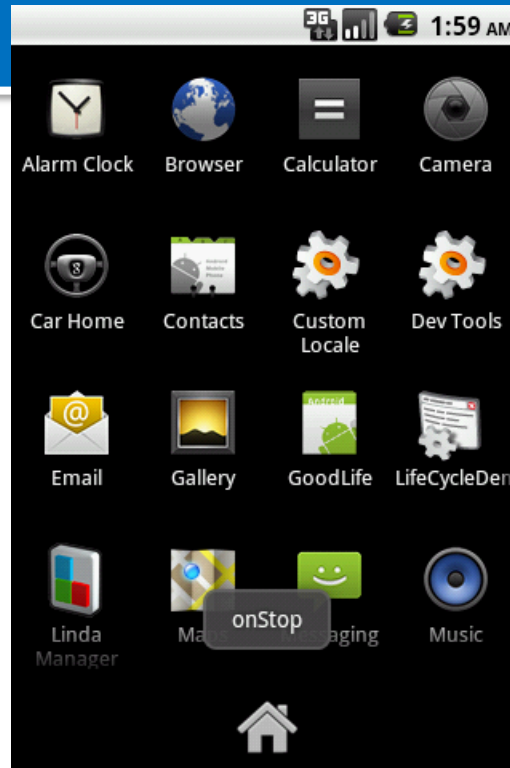
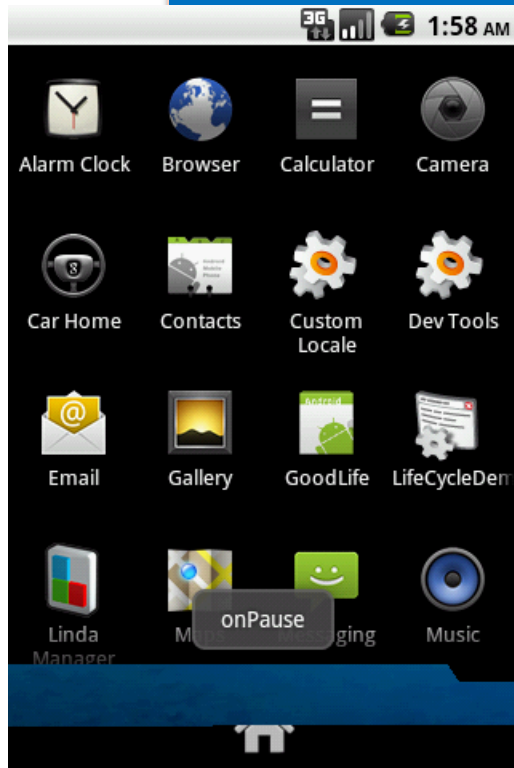


Example: Life Cycle

onPause...

onStop...

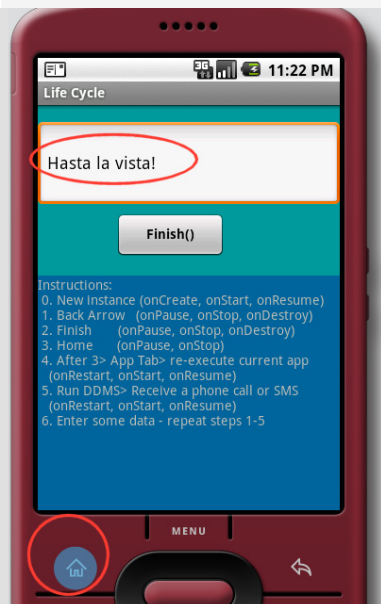
onDestroy...



After pressing "Back Arrow"

Example: Life Cycle

After pressing “Home”	After re-executing AndLife2	After “Back Arrow” or Finish
<p>onSaveInstanceState ></p> <p>onPause ></p> <p>onStop ></p>	<p>onRestart ></p> <p>onStart ></p> <p>onResume ></p>	<p>onPause ></p> <p>onStop ></p> <p>onDestroy ></p>



Preserving State Information

1. Enter data: “Hasta la vista!”
2. Click Home button
3. onSaveInstanceState > onPause > onStop
4. Read your SMS
5. Execute an instance of the application
6. onRestart > onStart > onResume
7. You see the data entered in step 1

End of Example

Life Cycle – QUESTIONS ?

Appendix

Using Bundles to Save State

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ... somevalue = savedInstanceState.getString(SOME_KEY);
    ...
}
...
@Override protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putString(SOME_KEY, "blah blah blah");
}
```

Bibliography:

1. Android Developers.
<http://developer.android.com/reference/android/app/Activity.html>
2. Professional Android Application Development by Reto Meier ISBN: 978-0-470-34471-2. Wrox Publications, 2008.
3. Unlocking Android by Frank Ableson, Charlie Collins, and Robi Sen. ISBN 978-1-933988-67-2. Manning Publications, 2009.
4. Professional Android 2 Application Development (Wrox Programmer to Programmer) by Reto Meier. ISBN-10: 0470565527. Wrox Pub. 2010.
5. The Busy Coder's Guide to Advanced Android Development by Mark Murphy. ISBN ISBN: 978-0-9816780-5-4. CommonsWare Pub. 2012.
6. Android Programming Tutorials by Mark Murphy. ISBN ISBN: 9 ISBN: 978-0-9816780-7-8. CommonsWare Pub. 2012.