CS 646 Android Mobile Application Development
Spring Semester, 2015
Doc 3 Android Basics
Jan 27, 2015

How all this works

R - Connection between resources & code

GUI Builder

res/layout/main.xml

```
<TextView
   android:id="@+id/countOutput"
   ...
```

Source Code

```
countOutput = (TextView)
     this.findViewById(R.id.countOutput);
```

R.java

```
public static final class id {
    public static final int clickButton=0x7f050001;
    public static final int countOutput=0x7f050000;
}
```

onCreate

```
public class ClickCountActivity extends Activity {
       TextView countOutput;
       int count = 0;

    @Override
     public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       countOutput = (TextView) this.findViewById(R.id.countOutput);
    }
```

When app is started ClickCountActivity is created and "onCreate" is called. The file res/layout/main.xml is read and the view described in it is creates. R.layout.main is a reference to that view object. In the file res/layout/main.xml one can give GUI elements ids. R.id.countOutput returns the GUI element with the id "countOutput". Android Studio will subclass ActionBarActivity instead of Activity.

layout magic

```
<Button
   android:id="@+id/clickButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/clickButtonLabel"
   android:onClick="increase" />
```

"@+id/clickButton" is how we give the id a name. The "@+" tells the compiler(?) to add this to the R.java file. "onClick" indicates the method to call when the button is clicked. The method has one argument of type view, which is the GUI element that generated the click.

Responding to the click

```
public class ClickCountActivity extends Activity {
      TextView countOutput;
      int count = 0;

   public void increase(View button) {
      Log.i("rew", "increase");
      count++;
      countOutput.setText(String.valueOf(count));
   }
```

increase is the method that the button will call when clicked (see previous slide). The argument is the button that caused the method to be called. In this case we only have one button so we do not use the argument. Log.i is a log statement. All log statements are sent to "LogCat" not the console. See next slide for more about logging. You may have go to DDMS and select the emulator for the output to show in "LogCat". System.out.print does work, but its output also goes "LogCat" not the console.

Logging

X (or levels)

v	Verbose
d	Debug
i	Info
w	Warning
e	Error
wtf	What a Terrible Failure Report condition that should not happen

Log.X(tag, message)
Log.X(tag, message, Exception)

Log file contains a lot of messages

Can filter based on
    Tags & Levels

Documentation states that debug log messages are stripped at runtime
That is false

Android Building Blocks

Basic Android Application Parts

- Activities
  - UI building block
  - Views & Activity subclasses
- Content Providers
  - Shares data between applications
- Intents
  - How your code starts a new activity
- Services
  - Long-running nonGUI code
- AndroidManifest.xml
- R.java
- layouts

- Fragments
  - Sub-activity UI container
  - Android 3.0+

Activity

Code that does some work

Single, focused thing that a user can do

Usually each screen(View) has its own activity

An application may have multiple screens, hence multiple activities

An application runs in its own Linux process

Activities can be viewless

Application

One or more screens (view)

Each screen has an activity

When go to new screen previous activity is stored on back stack

Back button

- Kills current activity
- Makes activity on top of back stack current

Home button

- Suspends current application
- Application and its activities just paused

Activity Life Cycle

Activity

Code that does some work

Single, focused thing that a user can do

Usually each screen(View) has its own activity

An application may have multiple screens, hence multiple activities

An application runs in its own Linux process

Activities can be viewless

Application

One or more screens (view)

Each screen has an activity

When go to new screen previous activity is stored on back stack

Back button

- Kills current activity
- Makes activity on top of back stack current

Home button

- Suspends current application
- Application and its activities just paused

Tasks

Sequence of activities the user follows to accomplish an objective

A user can

- Interrupt a task to start a new task
- Resume the first task where they left off

Tasks & Applications

Many applications are self contained

So task is sequence of activities from the application

Some applications use activities from other applications

Use phone
Show contacts
Use Web browser
Play music

So task is sequence of activities from multiple applications

Interrupting a Task

User presses Home and starts an application

Notifications

Activity Stack

Back Stack

History of activities used by user

May include activities of different applications

Back button

- Removes top of activity stack
- Makes next activity active

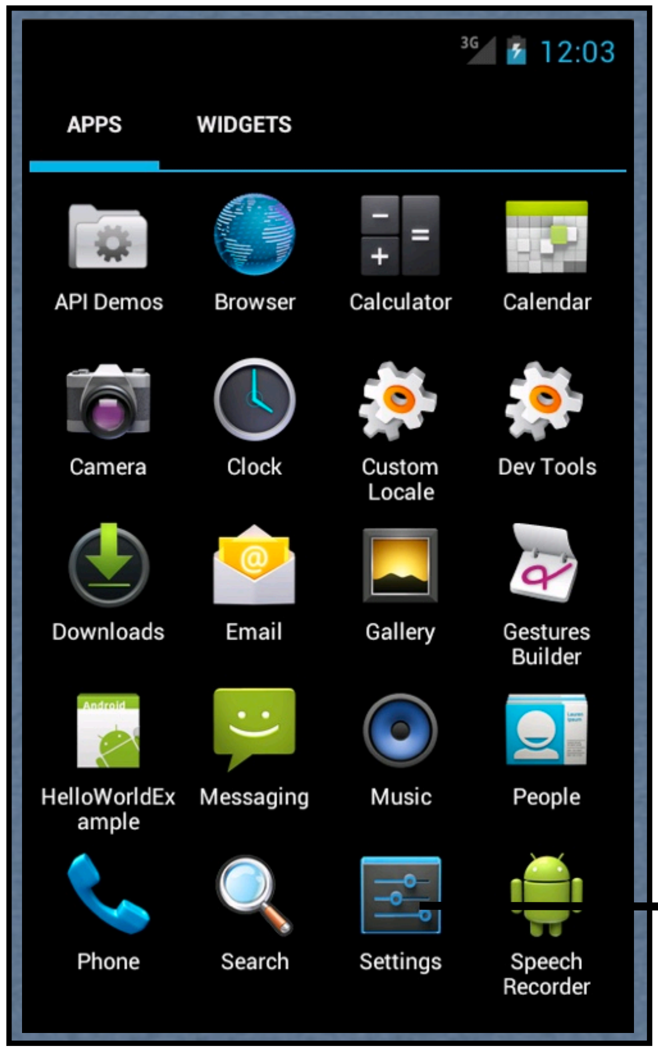
Home button

- Activity stack remains
- Starting another application starts new activity stack

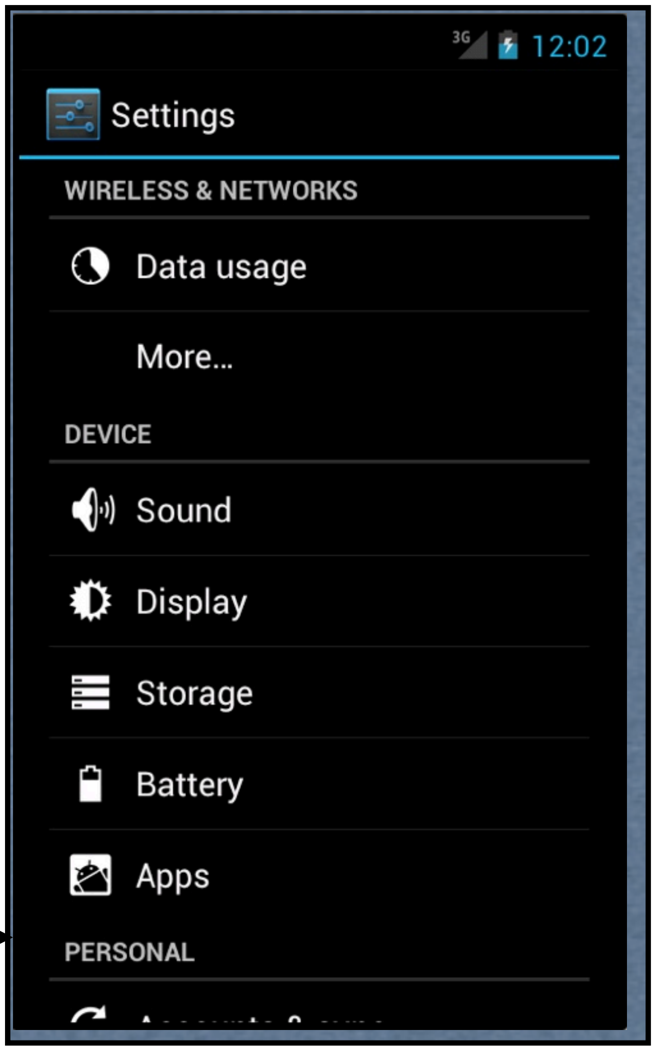
Stack only goes back to the start of the application at Home

Back Stack Example

Back Stack



Start Settings app

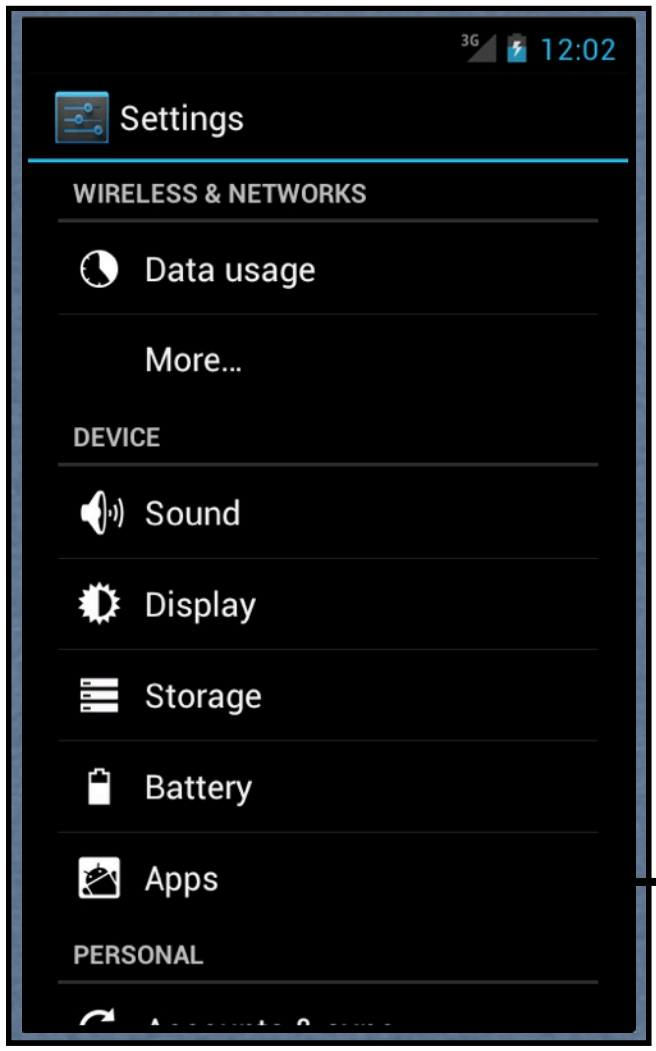


Back Stack Example

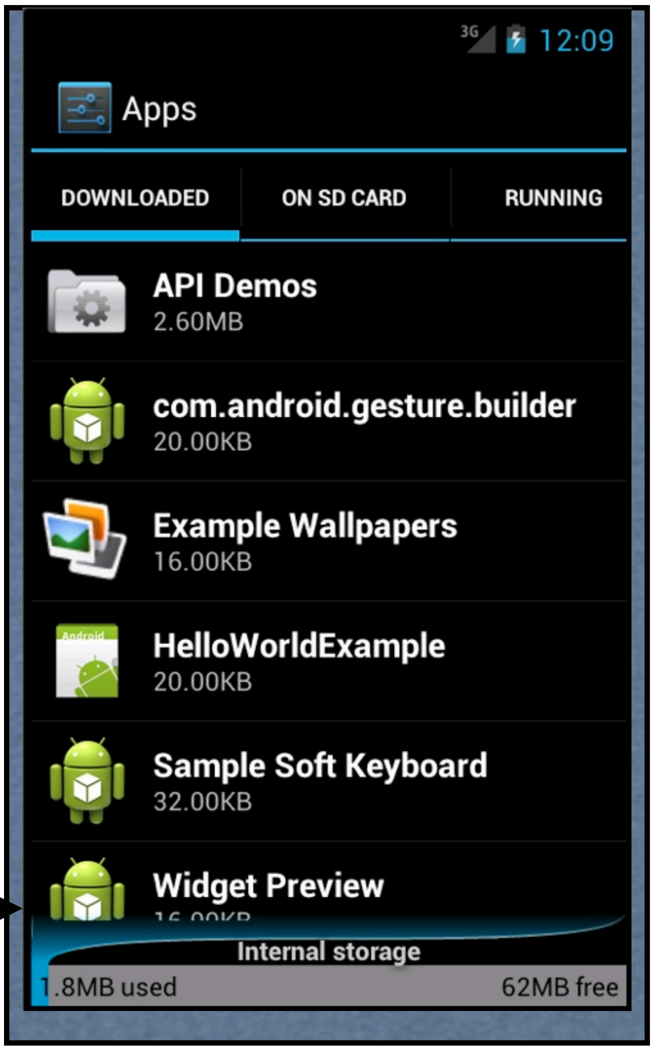
Settings

Back Stack

Apps
Settings



Apps activity

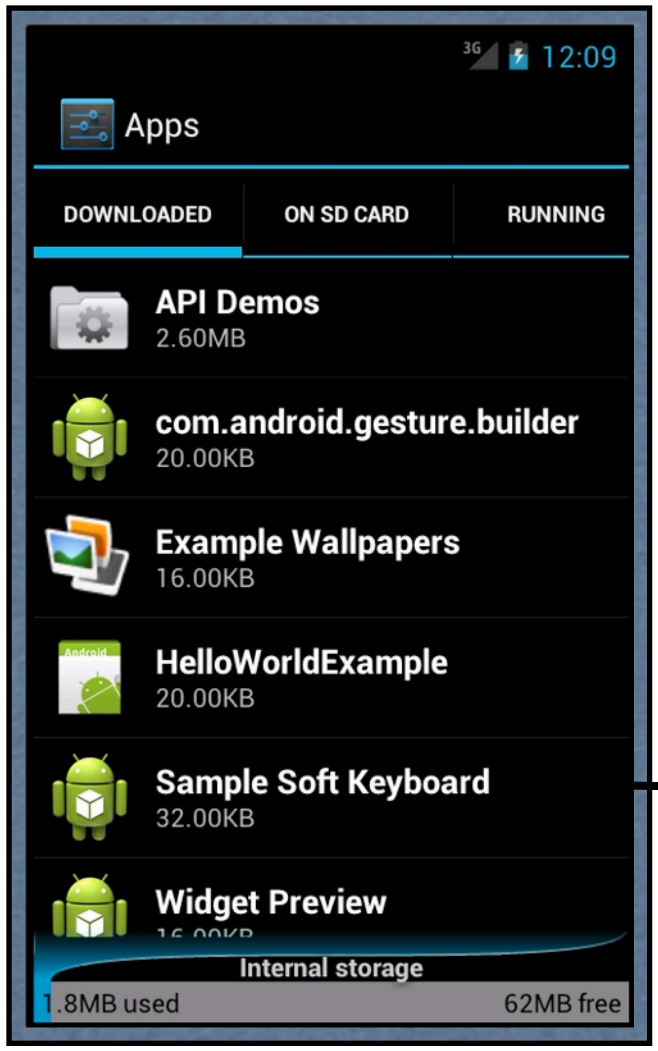


Back Stack Example

Apps
Settings

Back Stack

Soft Keyboard
Apps
Settings



Apps

DOWNLOADED | ON SD CARD | RUNNING

API Demos
2.60MB

com.android.gesture.builder
20.00KB

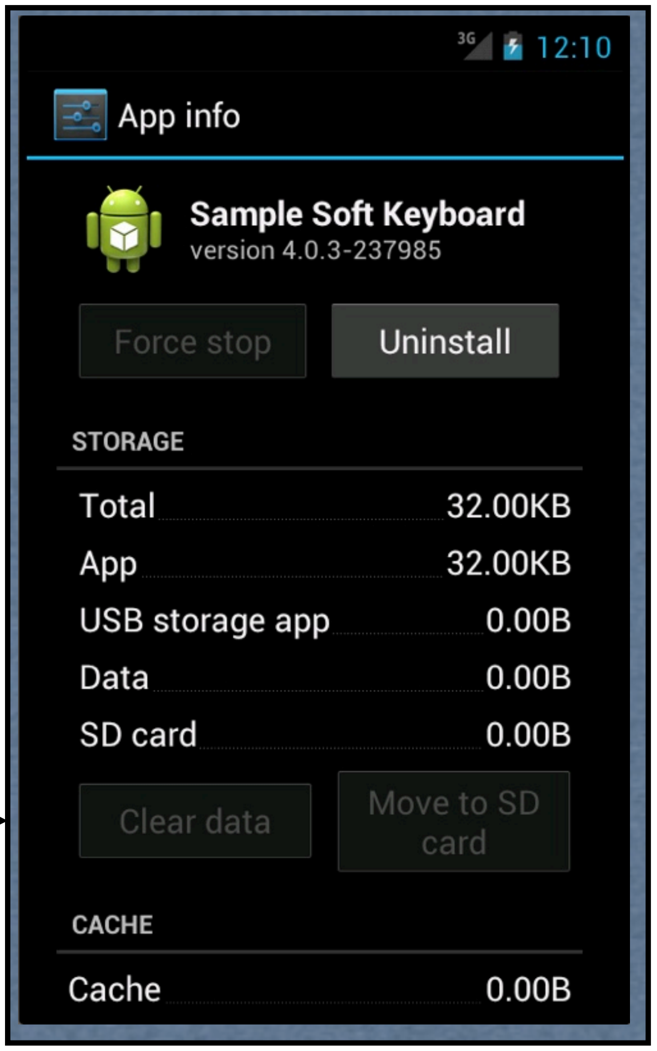
Example Wallpapers
16.00KB

HelloWorldExample
20.00KB

Sample Soft Keyboard
32.00KB

Widget Preview

Internal storage
1.8MB used 62MB free



App info

Sample Soft Keyboard
version 4.0.3-237985

Force stop | Uninstall

STORAGE

Total	32.00KB
App	32.00KB
USB storage app	0.00B
Data	0.00B
SD card	0.00B

Clear data | Move to SD card

CACHE

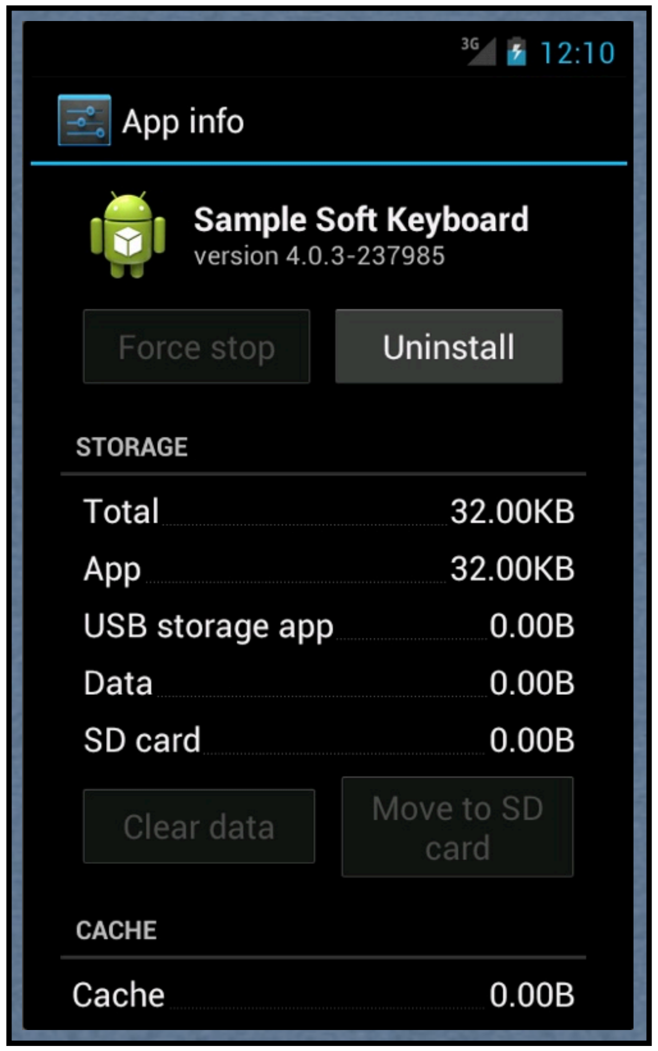
Cache	0.00B

Back Stack Example - Back Button

Soft Keyboard
Apps
Settings

Back Stack

Apps
Settings


App info

Sample Soft Keyboard
version 4.0.3-237985

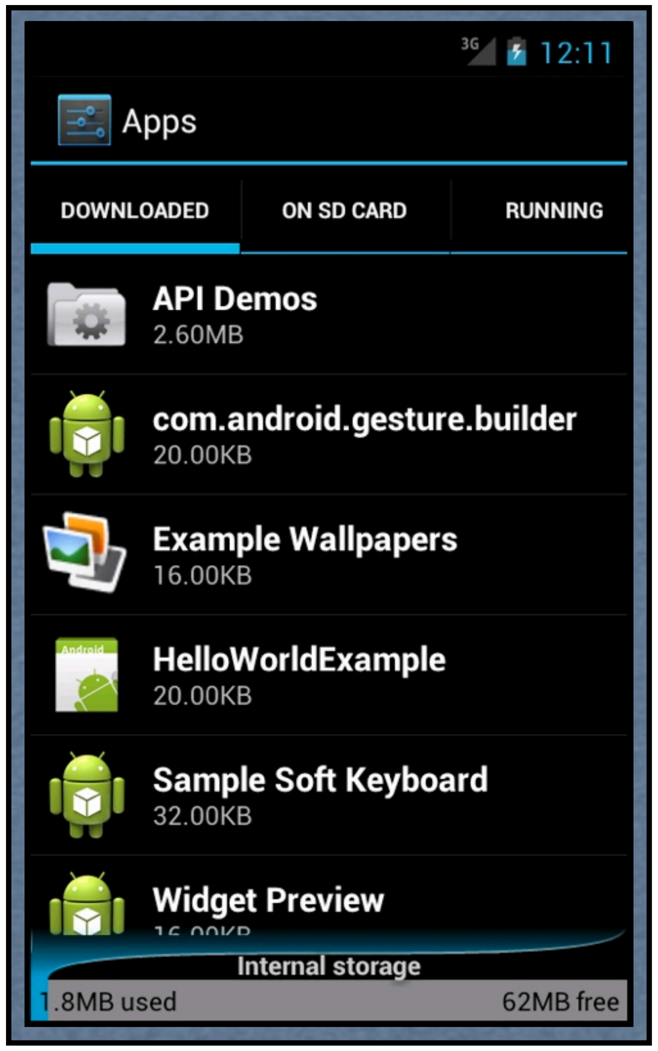
STORAGE

Total	32.00KB
App	32.00KB
USB storage app	0.00B
Data	0.00B
SD card	0.00B

CACHE

Cache	0.00B

Click back buttton


Apps

API Demos
2.60MB

com.android.gesture.builder
20.00KB

Example Wallpapers
16.00KB

HelloWorldExample
20.00KB

Sample Soft Keyboard
32.00KB

Widget Preview

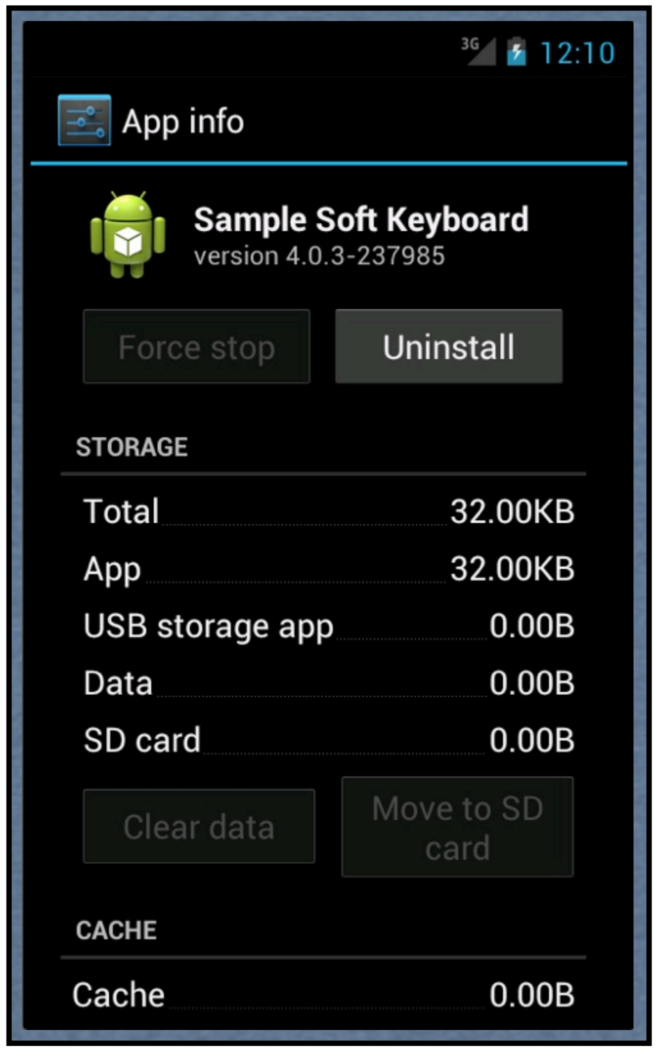
Internal storage
1.8MB used 62MB free

Back Stack Example - Home Button

Soft Keyboard
Apps
Settings

Back Stack

Soft Keyboard
Apps
Settings


12:10
App info
Sample Soft Keyboard
version 4.0.3-237985

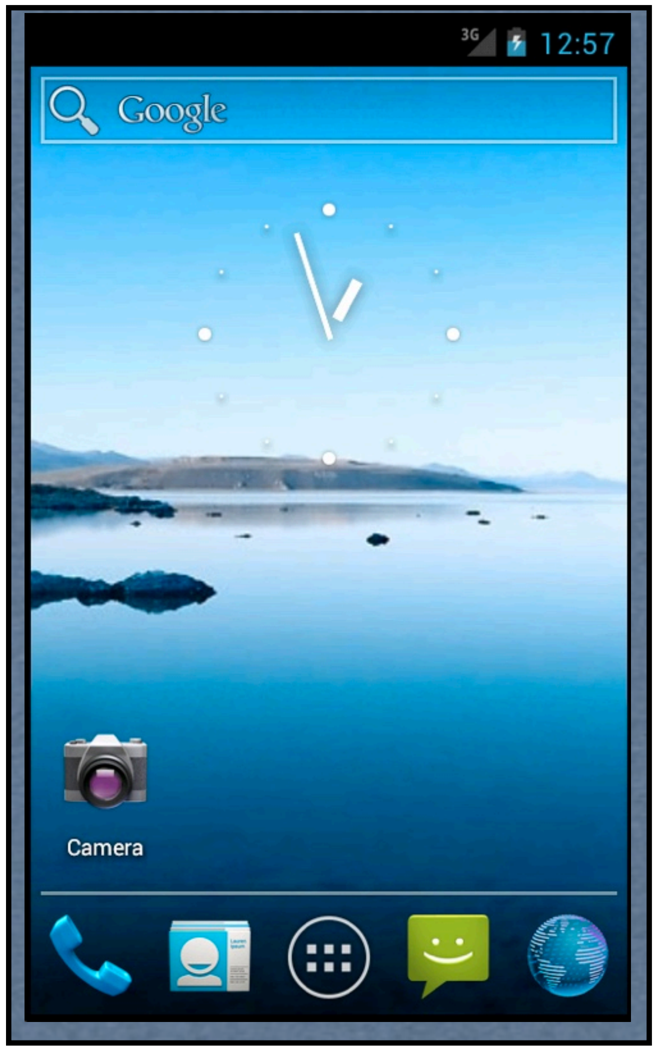
STORAGE

Total	32.00KB
App	32.00KB
USB storage app	0.00B
Data	0.00B
SD card	0.00B

CACHE

Cache	0.00B

Click home buttton



Tuesday, January 27, 15

The home button brings you back to the home screen. A new back stack will be created if we start a new app. The old back stack remains. If the user restarts the old app then activity on top of its back stack is used, the app does not start from beginning.

Applications & Activity Stacks

- Launching a non-running application
  - Create new activity stack
  - Put application's beginning activity on stack

- Launching a running application
  - Show activity on top of applications activity stack
  - That activity may be from another application

- Exceptions
  - Some background activities return to their initial screen
    - Contacts & Gallery

  - Some activities continue to run while in the background
    - Music player

See http://developer.android.com/guide/components/tasks-and-back-stack.html for a complete description

Activity Lifecycle States

Running (Resumed)

- Running activity in foreground of screen

Paused

- Lost focus, but still visible
- Retains all state information
- In extreme memory situations may be killed

Stopped

- Not visible
- Retains all state information
- Often will be killed

How activities can be killed

Kill the app

All activities in app back stack are killed

Back button

Current activity is killed

Lack of Memory

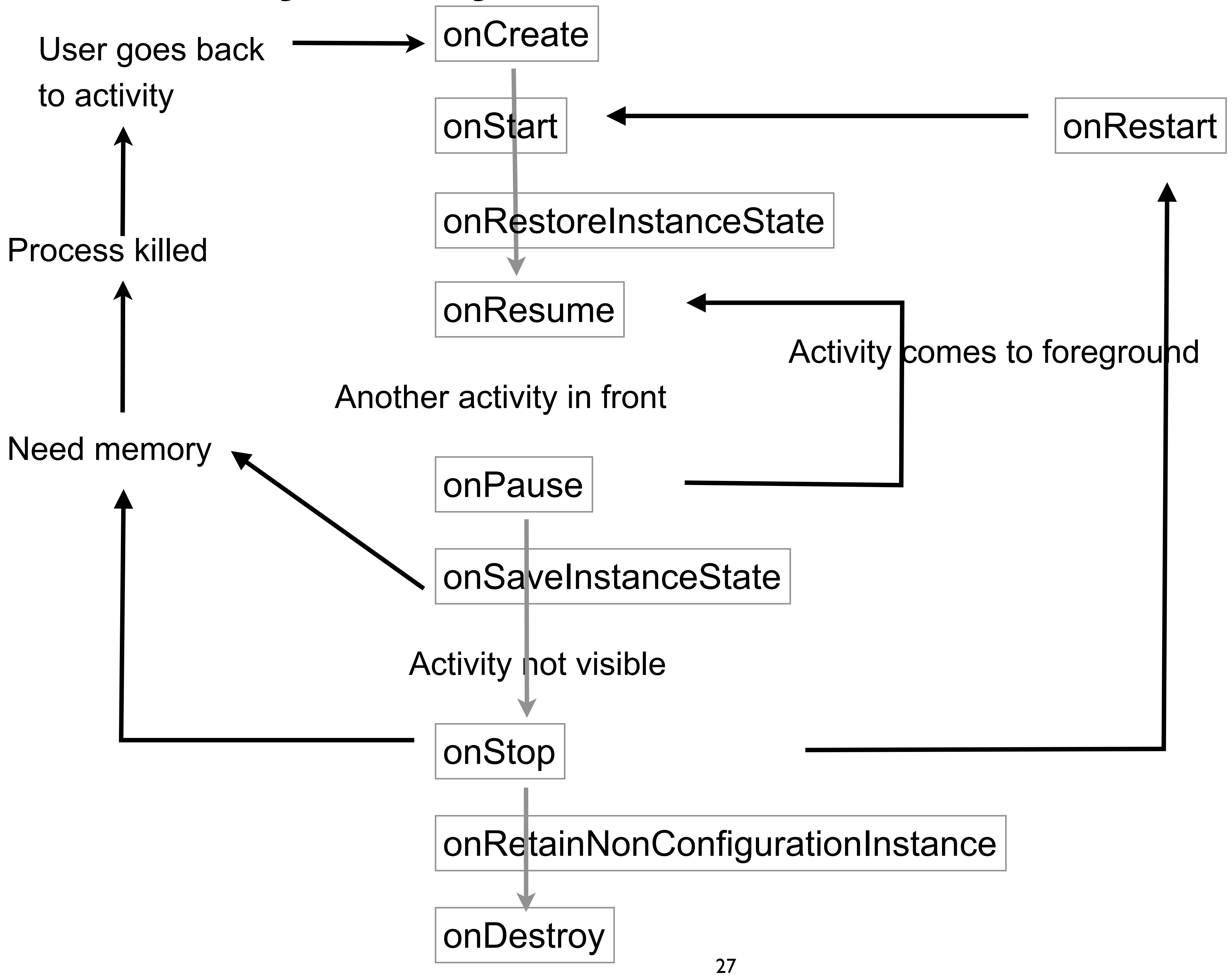
If run out of memory OS will kill activities in back stack

Device configuration changes

screen orientation, language change, keyboard availability(?), etc(?)

Activity is destroyed and recreated

Activity Life Cycle Methods



Important Issue

If OS kills activity in back stack to reclaim memory

We have to insure activity

- Looks and acts the same
- When user goes back to the activity

Saving State

When low on memory system will kill activities
- In activity stack
- Not visible

When user goes back to killed activity
- Activity must appear as it did before it was killed

Must save state of activity
- System will save state of views

Types of State to Save

Dynamic instance state

- State of instance variables of activity
- Needed so activity object can operate

Persistent state

- Information that should be available next time application is run
- Contact information in Address book

Overlap

- Persistent state is usually subset of dynamic state

Saving Persistent State

Do it in the onPause() method

- It will always be called
- One method that will always be called before activity is killed

onStop() and onDestroy() are not always called

onStop()

Called when activity is no longer visible

Not always called

Android 3.0 and later

onStop() will be called

Can save persistent in onStop()

onDestroy()

Used to free resources like threads

There are situations when

"system will simply kill the activity's hosting process
without calling this method"

Saving Data

- Temporarily
    - onSaveInstanceState
    - onRetainNonConfigurationInstance
- Permanently
    - Preferences
    - Files
        - Internal
        - External (SD card)
    - SQLite database
    - Content Providers
    - Network

Temporary Data

When

  App is destroyed and immediately recreated

  App is destroyed while in background due to low memory issue

Fields

Data in UI

Temporary Data

Save in onSaveInstanceState

Recover in
    onCreate(Bundle) or
    onRestoreInstanceState(Bundle)

Data in bundles
Base types (Int, etc)
    + Serializable
    + Parcelable

```
static final String FOO_KEY = "foo field";

protected void onSaveInstanceState(Bundle outState ) {
    super.onSaveInstanceState(outState);
    outState.putInt(FOO_KEY, foo);
}

protected void onRestoreInstanceState(Bundle savedInstanceState) {
    foo = savedInstanceState.getInt(FOO_KEY);
}
```

Issue - Data in UI

Call super.onSaveInstanceState() so Android can save the state of your UI elements

```
protected void onSaveInstanceState(Bundle outState ) {
    super.onSaveInstanceState(outState);
    outState.putInt(FOO_KEY, foo);
}
```

Issue - Not all data can be serialized

Bundles can only hold

- Base types (Int, etc)
  - + Serializable
  - + Parcelable

How do save data that bundle can not hold?

onRetainNonConfigurationInstance

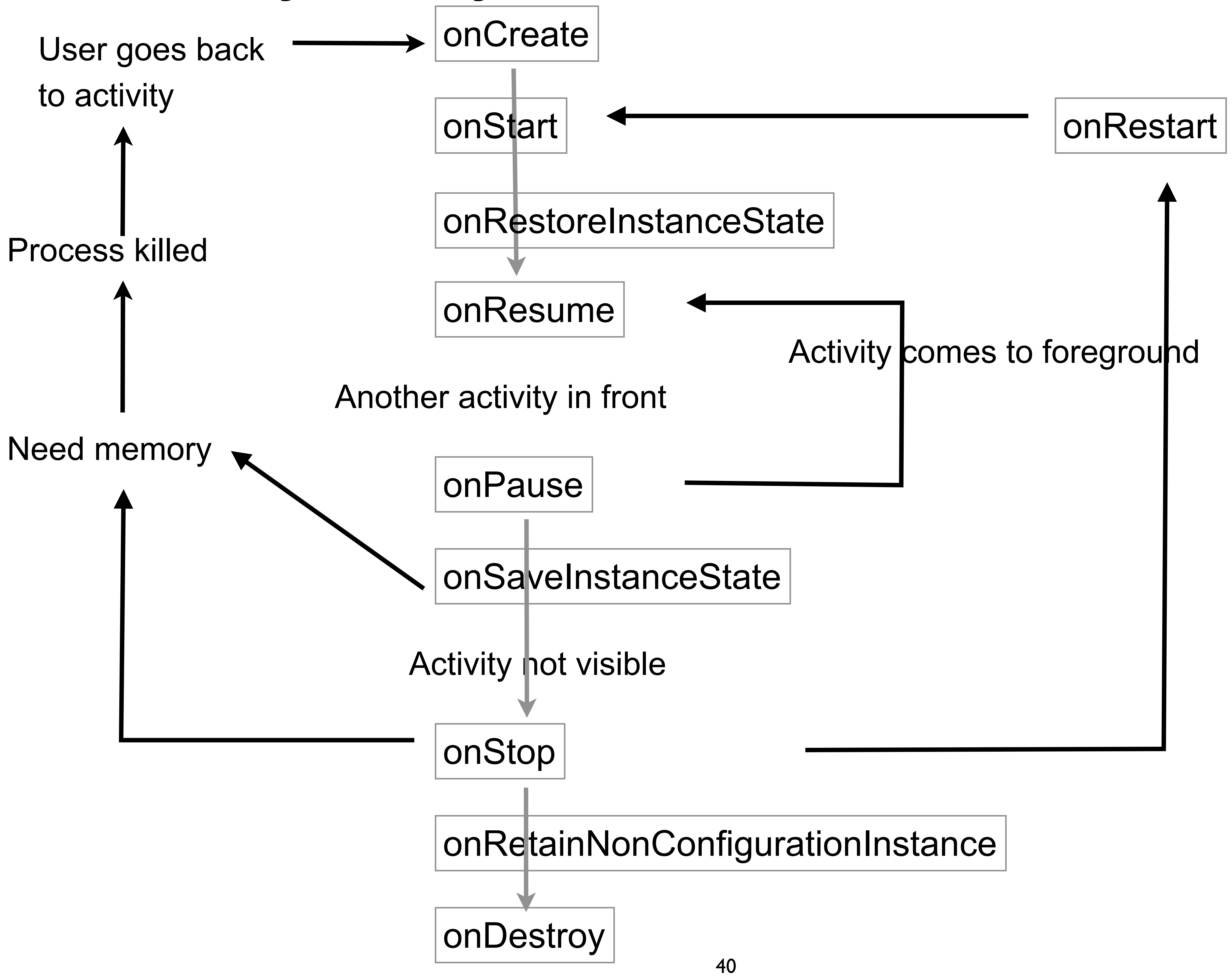
Only called if activity will be recreated immediately

```
public Object onRetainNonConfigurationInstance() {
        return new Integer(stopped);
}
```

Call getLastNonConfigurationInstance in onCreate or onStart to get saved value

```
protected void onStart() {
                super.onStart();
                Integer savedObject = (Integer) getLastNonConfigurationInstance();
                if (savedObject != null) {
                        stopped = savedObject.intValue();
                }
        }
```

Activity Life Cycle Methods



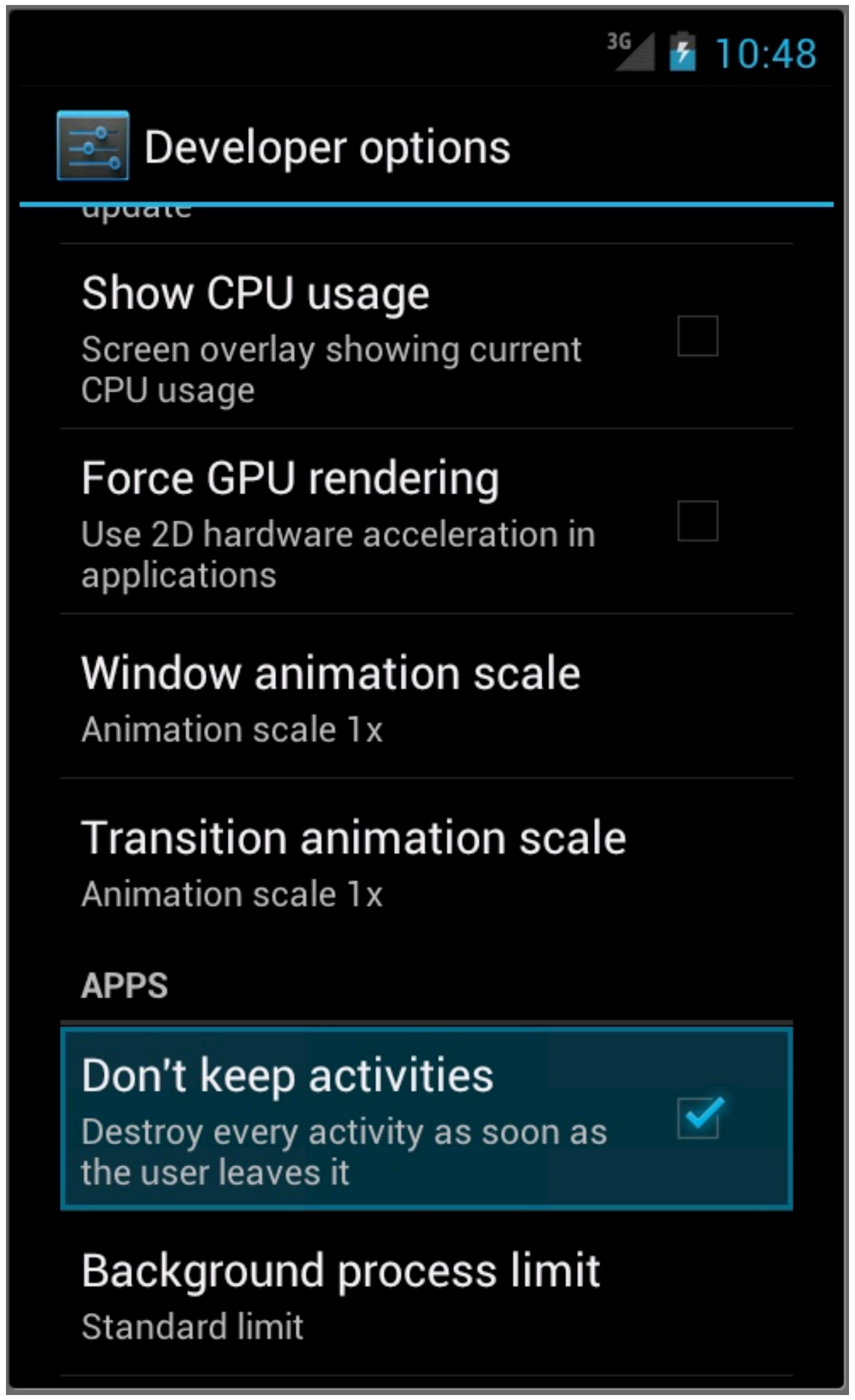
How Do We know it Works?

Some methods are only called when activity is killed

In Emulator

- System Settings (in Menu)
- Developer Options

Don't keep activities


10:48

Developer options

update

Show CPU usage
Screen overlay showing current CPU usage ☐

Force GPU rendering
Use 2D hardware acceleration in applications ☐

Window animation scale
Animation scale 1x

Transition animation scale
Animation scale 1x

APPS

Don't keep activities
Destroy every activity as soon as the user leaves it ✓

Background process limit
Standard limit

How Do We know it Works?

Rotate the device - activity is destroyed and recreated

Rotating Emulator

control - F11 or Keypad 7 - previous orientation
control - F12 or Keypad 9 - next orientation

On Mac

fn - control - F11 or Keypad 7 - previous orientation
fn - control - F12 or Keypad 9 - next orientation

Emulator Keyboard Command

http://developer.android.com/tools/help/emulator.html#controlling