Recording in Progress

This class is being recorded
Please turn off your video and/or video if you do not wish to be recorded
CMSC436: Programming Handheld Systems
The Activity Class
Today’s Topics

The Activity class
The Task Backstack
The Activity lifecycle
Starting an Activity
Handling configuration changes
The Activity Class

Provides a visual interface for user interaction
Each Activity typically supports one focused thing a user can do, such as
  Viewing an email message
  Showing a login screen
Activities and Application

Applications often comprise several Activities
User interaction results in navigating across these Activities
Android’s Navigation Support

Tasks
The Task Backstack
Suspending and resuming Activities
Tasks

A set of related Activities

These Activities can be from different applications

Most Tasks start at the home screen
Task Backstack

When an Activity is launched, it goes on top of the backstack

When the Activity is destroyed, it is popped off the backstack
The Activity Lifecycle

Activities are created, suspended, resumed and destroyed as necessary when an application executes.

Some of these actions depend on user behavior:
- e.g., User hits back button

Some depend on Android:
- e.g., Android can kill Activities when it needs their resources
Activity Lifecycle States

Resumed/Running—Visible, user interacting
Paused—Visible, user not interacting, can be terminated (in older versions of Android)
Stopped—Not visible, can be terminated
The Activity Lifecycle Methods

Android announces Activity lifecycle state changes to Activities by calling specific Activity methods
Some Activity Callback Methods

```kotlin
protected open fun onCreate(savedInstanceState: Bundle?): Unit
protected open fun onStart(): Unit
protected open fun onResume(): Unit
protected open fun onPause(): Unit
protected open fun onRestart(): Unit
protected open fun onStop(): Unit
protected open fun onDestroy(): Unit
```
The Activity Lifecycle

- Activity Launched
  - onCreate()
  - onStart()
    - onResume()
    - onPause()
    - onStop()
    - onDestroy()
- Activity Shut Down
- onRestart()
The Activity Lifecycle

- Activity Launched
  - onCreate()
  - onStart()
    - onResume()
      - onPause()
      - onStop()
        - onDestroy()
          - Activity Shut Down
  - onRestart()
The Activity Lifecycle

Activity Launched
- onCreate()
  - onStart()
    - onResume()
      - onPause()
        - onStop()
          - onDestroy()
            - Activity Shut Down

Entire Lifetime

App. Proc. Killed
The Activity Lifecycle
The Activity Lifecycle

- Activity Launched
  - onCreate()
    - onStart()
      - onResume()
        - onPause()
          - onStop()
            - onDestroy()
              - Activity Shut Down

  - Activity Running

- Visible & in Foreground
MapLocation

1600 Pennsylvania Avenue NW Washington DC 20500

SHOW MAP
The Activity Lifecycle: MapLocation

- Activity Launched
  - onCreate()
  - onStart()
  - onResume()
    - Activity Running
      - onPause()
      - onStop()
      - onDestroy()
  - onRestart()
- Activity Shut Down
The Activity Lifecycle: MapLocation
The Activity Lifecycle: MapLocation
**onCreate()**

Called when Activity is created
Sets up initial state

Call `super.onCreate()`
Set the Activity’s content view
Retain references to UI views as necessary
Configure views as necessary
class MapLocation : Activity() {
    companion object {
        const val TAG = "MapLocation"
    }

    // UI elements
    private lateinit var addrText: EditText
    private lateinit var button: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        /* Required call through to Activity.onCreate()
        Restore any saved instance state, if necessary */
        super.onCreate(savedInstanceState)

        // Set content view
        setContentView(R.layout.main)
    }
MapLocation.kt

// Initialize UI elements
addrText = findViewById(R.id.location)
button = findViewById(R.id.mapButton)
// Link UI elements to actions in code
button.setOnClickListener { processClick() }

// Called when user clicks the Show Map button
private fun processClick() {
    try {
        // Process text for network transmission
        var address = addrText.text.toString()
        address = address.replace(' ', '+')

        // Create Intent object for starting Google Maps application
        val geoIntent = Intent(Intent.ACTION_VIEW, Uri.parse("geo:0,0?q=$address"))
    } catch (e: Exception) {
        // Handle error
    }
}
MapLocation.kt

```kotlin
geoIntent.resolveActivity(packageManager)?.let {
    // Use the Intent to start Google Maps application using
    // Activity.startActivity()
    startActivity(geoIntent)
}

} catch (e: Exception) {
    // Log any error messages to LogCat using Log.e()
    Log.e(TAG, e.toString())
}
```
onStart()

Activity is about to become visible

Typical actions

Start visible-only behaviors

Loading persistent application state
onResume()

Activity is visible and about to start interacting with user

Typical actions

Start foreground-only behaviors
onPause()

Focus about to switch to another Activity

Typical actions

  Shutdown foreground-only behaviors
  Save persistent state
**onStop()**

Activity is no longer visible to user
  may be restarted later

Typical actions
  Save persistent state
  Do CPU-intensive save procedures

Note: Pre-Honeycomb - this method may not be called if Android kills your application
onRestart()

Called if the Activity has been stopped and is about to be started again

Typical actions

   Special processing needed only after having been stopped
onDestroy()

Activity is about to be destroyed

Typical actions

Release Activity-wide resources

Note: may not be called if Android kills your application
override fun onStart() {
    super.onStart()
    Log.i(TAG, "The activity is visible and about to be started.")
}

override fun onRestart() {
    super.onRestart()
    Log.i(TAG, "The activity is visible and about to be restarted.")
}

override fun onResume() {
    super.onResume()
    Log.i(TAG,
    "The activity is visible and has focus (it is now "resumed")")
}
override fun onPause() {
    super.onPause()
    Log.i(TAG, "Another activity is taking focus (this activity is about to be \"paused\")")
}

override fun onStop() {
    super.onStop()
    Log.i(TAG, "The activity is no longer visible (it is now \"stopped\")")
}

override fun onDestroy() {
    super.onDestroy()
    Log.i(TAG, "The activity is about to be destroyed.")
}
}
Starting Activities

Create an Intent object matching the Activity to start
Starting Activities

Pass newly created Intent to methods, such as:

startActivity()

startActivityForResult()

Invokes a callback method, onActivityResult(), when the called Activity finishes to return a result to the calling Activity.
// Called when user clicks the Show Map button
private fun processClick() {
    try {
        ... // Create Intent object for starting Google Maps application
        val geoIntent = Intent(Intent.ACTION_VIEW,
                                Uri.parse("geo:0,0?q=$address"))

        geoIntent.resolveActivity(packageManager)?.let {
            // Use the Intent to start Google Maps application using
            // Activity.startActivity()
            startActivity(geoIntent)
        }
    }
}
MapLocationFromContacts

Similar to MapLocation, but gets address from Contacts database
private fun startContactsApp() {

    // Create Intent object for picking data from
    // Contacts database
    val intent = Intent(Intent.ACTION_PICK)
    intent.type = CONTENT_ITEM_TYPE

    intent.resolveActivity(packageManager)?.let {
        // Use intent to start Contacts application
        // Variable PICK_CONTACT_REQUEST identifies this operation
        startActivityForResult(intent, PICK_CONTACT_REQUEST)
    }
}
Activity.setResult()

The started Activity can set its result by calling Activity.setResult()

fun setResult(resultCode: Int): Unit

fun setResult(resultCode: Int, data: Intent!): Unit
Activity.setResult()

resultCode - an Int

    RESULT_CANCELED
    RESULT_OK
    RESULT_FIRST_USER

    Custom resultCodes can be added
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent) {

    if (resultCode == RESULT_OK && requestCode == PICK_CONTACT_REQUEST) {
        ...
        if (null != formattedAddress) {
            ...
            // Create Intent object for starting Google Maps application
            val geoIntent = Intent(Intent.ACTION_VIEW,
                Uri.parse("geo:0,0?q=${formattedAddress}"))
            // Use the Intent to start Google Maps application using
            // Activity.startActivity()
            startActivity(geoIntent)
        }
    }
    ...
}
Configuration Changes

Keyboard, orientation, locale, etc.

Device configuration can change at runtime

On configuration changes, Android usually kills the current Activity & then restarts it
Configuration Changes

Activity restarting should be fast

Options

- Save Activity state in Bundle
- Retain a separate Object
- Manually handle the configuration change (not usually recommended)
Saving Activity State

Android saves some information such as View state in a Bundle

You must save other state yourself
Saving Activity State

Android calls onSaveInstanceState(Bundle)
  after onStop() for API 28+
  before onStop() for API <28
Save Activity instance state to system-provided Bundle
Saving Activity State

When Activity is restarted, you can restore Activity state from a system-provided Bundle in:

- `onCreate(Bundle)`
- `onRestoreInstanceState(Bundle)`, which is called between `onStart()` and `onPostCreate()`
class TickerDisplayActivity : Activity() {
    companion object {
        private const val COUNTER_KEY = "COUNTER_KEY"
        private const val DELAY: Long = 1000
    }

    private lateinit var mCounterView: TextView
    private lateinit var mUpdater: Runnable
    private var mCounter = 0
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    setContentView(R.layout.activityTickerDisplay)

    mCounterView = findViewById(R.id.counter)

    savedInstanceState?.let { mCounter = it.getInt(COUNTER_KEY) }

    ...
}
// Save instance state
public override fun onSaveInstanceState(bundle: Bundle) {

    // Save mCounter value
    bundle.putInt(COUNTER_KEY, mCounter)

    // call superclass to save any view hierarchy
    super.onSaveInstanceState(bundle)
}
Retaining an Object

Hard to recompute data can be cached to speed up handling of configuration changes

Current recommendation is to store state in a Fragment

We’ll come back to this in a later lesson
Manual Reconfiguration

Can prevent system from restarting Activity
Declare the configuration changes your Activity handles in AndroidManifest.xml file, e.g.,
<activity android:name=".MyActivity"
    android:configChanges="orientation|screensize|keyboardHidden"...>
Manual Reconfiguration

When configuration changes, Activity’s onConfigurationChanged() method is called

Passed a Configuration object specifying the new device configuration
Manual Reconfiguration Caveat

Should generally avoid manual approach

- Hard to get right
- Fragile to system changes
Next

The Intent Class
Example Applications

MapLocation
MapLocationFromContacts
Ticker