Mobile Application Design and Development

CS 5520

Adrienne Slaughter

Fall 2021

Northeastern University
College of Computer and Information Science

Today's Schedule

- Review A4/Clean Android Architecture

Objectives

- Threads & Processes
- Broadcast Receivers
  - Short excursion into Intents & Intent-filters
- Services

Threads & Asynchronous Tasks

- What is a thread?

The Problem:

- We want to do something in another thread.

Application Lifecycle

- Main Thread == UI Thread
- Android manages resources by killing processes that have lower priority when resources are scarce
- Priorities:
  - Foreground process
    - Required for what the user is doing
  - Visible process
    - User is aware of what it's doing
  - Service process
    - Service; Not visible to user, but is usually something the user cares about.
  - Cached process
    - Not currently active/needed

SUPER IMPORTANT

- Two main rules of threads in Android development:
  - Do not block the UI thread
  - Do not access the Android UI toolkit from outside the UI thread
- UI Thread:
  - Draw something on the screen
  - Listen for inputs from user
  - Deliver those inputs to code (does something in response)

Example

What is an Asynchronous Task?

- AsyncTask enables proper and easy use of the UI thread.
- allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

```
private class DownloadFilesTask extends AsyncTask<Params, Progress, Result> {
    protected <ResultType> doInBackground(Params... time) {

    }

    protected void onProgressUpdate(<ProgressType>... progress) {
        setProgressPercent(progress[0]);
    }
}
```

When to use an Asynchronous Task?

- Anything that will block the UI thread.
- Examples:
  - Downloading data from the internet
  - Processing
- Can update the UI thread

SUMMARY: Creating an Async Task

- Create a class that extends AsyncTask
- Override the doInBackground() method
- Create an instance and call execute()
- Be sure to update the UI:
  - During execution via publishProgress() and onProgressUpdate()
  - After execution via onPostExecute()

The “Post” option

```
timerTV.post(new Runnable() {
    public void run() {
            timerTV.setText(Integer.toString(time));
    }
});
```

Broadcast Receivers

- Allows you to register to be notified of a system or application event
- Register for standard Android broadcast
  - Example system broadcasts: screen has turned off, the battery is low, user is present using phone, or a picture was captured.
- Applications can initiate broadcasts—e.g., to let other applications know that some data has been downloaded to the device and is available for them to use.
  - They don't display a UI, but can create a status bar notification to alert the user when a broadcast event occurs.

BroadcastReceiver

Usually, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

Important: you must complete tasks in a BroadcastReceiver in <10s. If you have a task that will take longer, you **must** start a new thread to avoid app assassin OS.

Northeastern University
College of Computer and Information Science

Example?

- Do live today or see video on your own.

For Reference:

- `./adb shell`
  - `am broadcast -a android.intent.action.CAMERA_BUTTON`
- List of actions:
  - `~/Library/Android/sdk/platforms/android-22/data/broadcast_actions.txt`

Summary: Creating a BroadcastReceiver

- Create a class that extends BroadcastReceiver
- Register the new BroadcastReceiver in the AndroidManifest
- Indicate which intent the BroadcastReceiver responds to
- Test by using the adb

Intents

- Composed of:
  - Component name
  - Action
  - Data
  - Category
  - Extras
  - Flags

Intent Filters

- Specifies what kind of actions, data and category of intents your application can handle
- Add a filter to a specific application component in your AndroidManifest.xml
- Android (the system) uses that to deliver intents to your application
- Complete lists found in: `~/Library/Android/sdk/platforms/android-22/data`
  (the data file of a specific installed Android SDK)

What is a service?

- Can perform long-running operations in the background
- Does not provide a user interface
- Started by other application components
- Can run in the background even if the user switches to another app
- Can be bound to for inter-process communication & further interaction

Service versus Thread

- Service allows running in the background, even if the user isn’t interacting with the application.
- Thread only runs as long as the user is interacting with the application.
- Both are blocking, so if there is a long-running task, do it in a separate thread.

When to use a service:

- Repetitive and potentially long running tasks:
  - Internet downloads, checking for new data, data processing, updating content providers

Example

Keep your Services secure

- To ensure that your app is secure, always use an explicit intent when starting a Service and do not declare intent filters for your services.
  - Using an implicit intent to start a service is a security hazard because you can't be certain what service will respond to the intent, and the user can't see which service starts.
- When starting a Service, *always specify the component name.*
  - Otherwise, you cannot be certain what service will respond to the intent, and the user cannot see which service starts.

Services and background processing

- By default, a service runs in the same process as the main thread of the application.
- Therefore, you need to use asynchronous processing in the service to perform resource intensive tasks in the background.
  - A commonly used pattern for a service implementation is to create and run a new Thread in the service to perform the processing in the background and then to terminate the service once it has finished the processing.

Summary: Creating a Service

- Create your Service class that extends Service
- Define your Service in the AndroidManifest
  - Do not put an intent-filter on your Service
- Create an Intent with your Service class name
- Call startService with the intent
- Make sure your service stops itself when complete

IntentService

- Similar to a service, but started with an Intent
- Runs on a separate thread (not the main thread)
- Can’t run tasks in parallel:
  - Get put into a queue and executed sequentially
- Stops itself

	Service	Thread	IntentService	AsyncTask
When to use ?	Task with no UI, but shouldn't be too long. Use threads within service for long tasks.	- Long task in general. - For tasks in parallel use Multiple threads (traditional mechanisms)	- Long task **usually** with no communication to main thread. **(Update)**- If communication is required, can use main thread handler or broadcast intents[3] - When callbacks are needed (Intent triggered tasks).	- Relatively long task (UI thread blocking) with a need to communicate with main thread.[3] - For tasks in parallel use multiple instances OR Executor [1]
Trigger	Call to method onStartService()	Thread start() method	Intent	Call to method execute()
Triggered From (thread)	Any thread	Any Thread	Main Thread (Intent is received on main thread and then worker thread is spawed)	Main Thread
Runs On (thread)	Main Thread	Its own thread	Separate worker thread	Worker thread. However, Main thread methods may be invoked in between to publish progress.
Limitations / Drawbacks	May block main thread	- Manual thread management - Code may become difficult to read	- Cannot run tasks in parallel. - Multiple intents are queued on the same worker thread.	- one instance can only be executed once (hence cannot run in a loop) [2] - Must be created and executed from the Main thread

http://techtej.blogspot.com.es/2011/03/android-thread-constructspart-4.html

Alarms & Alarm Manager

- An alarm can be used when you need your app to do something at a particular time.
- They let you fire Intents at set times and/or intervals.
- You can use them in conjunction with broadcast receivers to start services and perform other operations.
- They operate outside of your application, so you can use them to trigger events or actions even when your app is not running, and even if the device itself is asleep.
- They help you to minimize your app's resource requirements. You can schedule operations without relying on timers or continuously running background services.

Android Application Components

- Activities
- **Services**
- **Broadcast Receivers**
- Content providers
- Intents
- Widgets
- Notifications