Platform-Based Development: Background Processing

BS UNI studies, Spring 2019/2020

Dr Veljko Pejović
Veljko.Pejovic@fri.uni-lj.si

University *of Ljubljana*
Faculty *of Computer and Information Science*

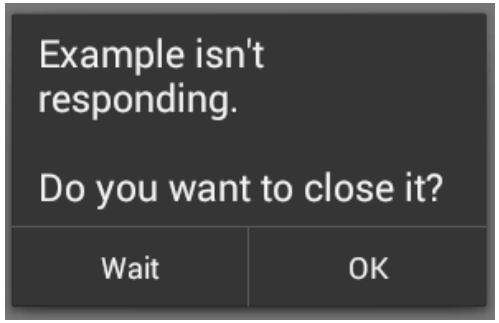
Concurrency in Android

- Java Threads
  - The most general method
- Service
  - Runs without the UI, however, by default on the same thread as the UI
- IntentService
  - Background work on a separate thread, to contact the UI use local broadcast
- AsyncTask
  - Background work on a separate thread, but with a tight integration with the UI

Threads

- A UI (Main) Thread is created and started when an application is launched
  - Listens for events on UI components
  - Loops infinitely
- Your code (by default) runs on the UI Thread
- Intensive work (database access, networking) can prevent the UI thread from processing UI interaction tasks

There are no separate threads for components



Threads

- Instead, run heavy/slow operations in background threads
- Background thread processing supported by:
  - Threads + Handlers
  - IntentService
  - AsyncTask
  - Thread + Service
- Different abstractions for different goals, e.g.:
  - A music player that runs in the background
  - An online social network post button

Thread and Handlers

- Java Threads + a Handler that enables communication among the threads
- A straightforward solution:
  - Create a worker Thread
  - Put an infinite loop in it and listen for new tasks
  - De-queue the tasks, for each task:
    - Execute
    - Report results back to the UI Thread via a Handler
  - Break the loop to kill the thread

Thread and Handlers Example

Looper, Message Queue, Handler

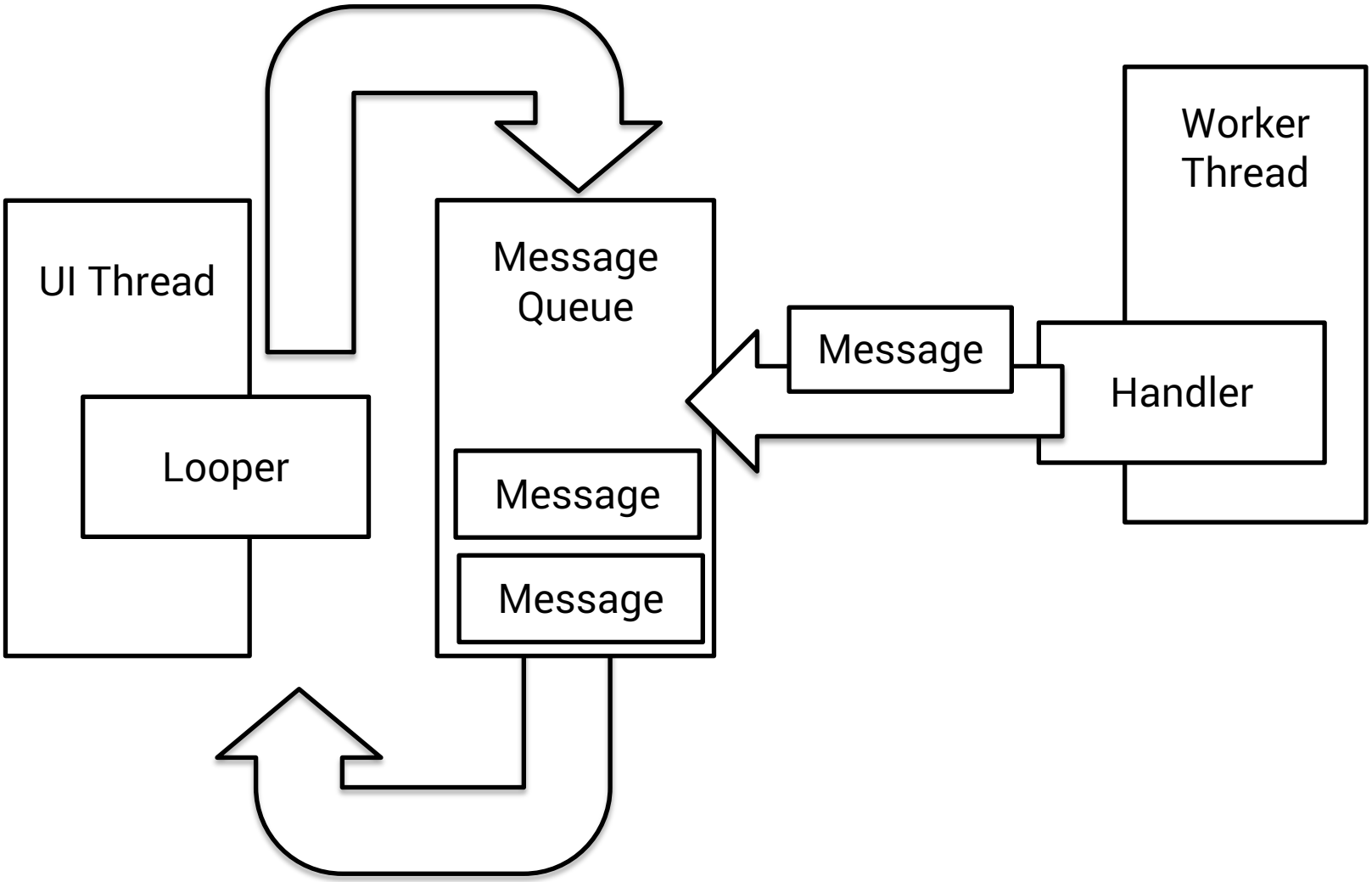
- Looper keeps the Thread alive in an infinite loop
  - Automatically created for the UI Thread
  - For custom threads, create it yourself or use HandlerThread
    - `Looper.prepare();`
    - `Looper.loop();`
    - `Looper.quit();`
- MessageQueue holds Messages/Runnables for a Thread
  - Message – for passing data to a thread
  - Runnable – a task that is executed when the thread is free (or after a predefined delay)

Looper, Message Queue, Handler

- Handler
  - Associated with a particular Thread (e.g. UI Thread)
  - Allows you to send Messages/Runnables to the MessageQueue and process them
  - Post a Message/Runnable immediately via sendMessage(Message m)/post(Runnable r) or after a delay via postDelayed(Runnable r, int msDelay)

```
new Handler(Looper.getMainLooper())
    .post(new Runnable() {
          @Override
          public void run() {
            // this will run in the main thread
          }
});
```

Thread and Handlers



HandlerThread

- A Thread that has a Looper
- Example use:
  - Instantiate a HandlerThread
  - Attach a Handler to your thread

```
HandlerThread handlerThread =
      new HandlerThread("MyHandlerThread");
handlerThread.start();
Looper looper = handlerThread.getLooper();
Handler handler = new Handler(looper);
```

Call .quit() to shut the thread down

HandlerThread Example

Services

- Activities run on the UI (main) thread and have a UI attached (layout)
  - Processing-heavy functions on the main thread impact the responsiveness
- Services can run on either the main or separate threads and do not have a UI attached
  - Run outside UI, for long-running operations
- Services are often more convenient than custom Threads for tasks than need to be “independent” and run even when the Activity is destroyed

Background and Foreground Service

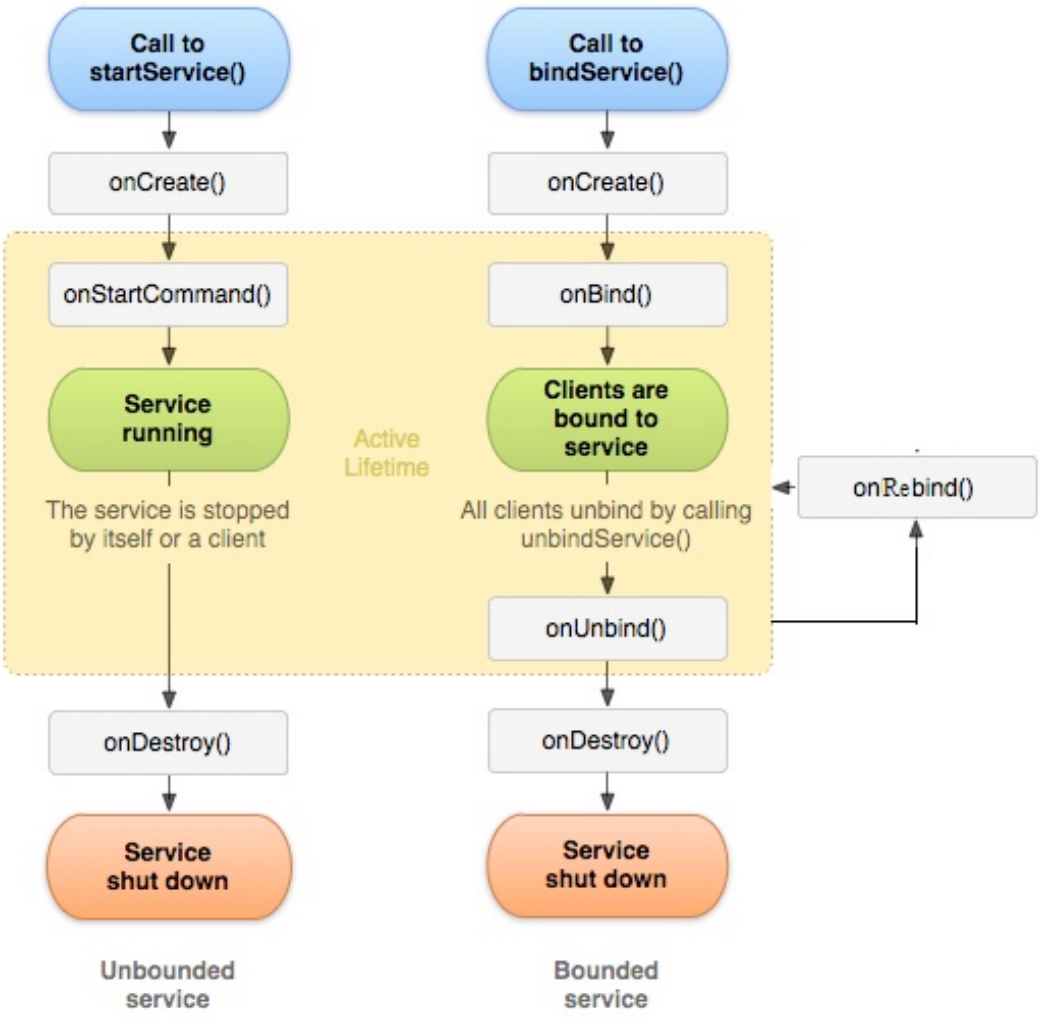
- Background Service
  - For actions that do not have to be noticed by the user (e.g. sensing a user's physical activity)
- Foreground Service
  - For actions that the user needs be aware of and should the control of (e.g. a music player app)
  - A foreground service must show a notification in the notification bar

Starting/Stopping a Service

- Services can be created:
  - Explicitly using Context.startService()
  - Implicitly, if not already running, when a client requests connection to a Service via Context.bindService()
- Services can be stopped:
  - From within the Service with stopSelf()
  - From another component with Context.stopService()

Services

- Multiple startService calls do not nest – you only have one service; however, onStartCommand() will be called repeatedly
- Service will be stopped only once with Context.stopService() or stopSelf()



Services – Bound

- Bound Services – like servers in a client-server paradigm
- Services started through binding, do not call onStartCommand()
- Return IBinder object from onBind(Intent) so that connected clients can call the Service
- The service remains running as long as the connection is established

Broadcast

- Messages sent from other components of your app, other apps or from the Android system
- Messages are wrapped in Intents

```
Intent intent = new Intent();
intent.setAction(ACTION);
intent.putExtra(STOP_SERVICE_BROADCAST_KEY, RQS_STOP_SERVICE);
sendBroadcast(intent);
```

- Send broadcasts
  - System sends certain broadcasts when an event happens, e.g. ACTION_BOOT_COMPLETED
  - Send custom broadcasts via sendBroadcast()

Broadcast

- Broadcasts are captured in an app/component if a BroadcastReceiver is registered in the code:
  - Create a BroadcastReceiver and impl. onReceive()

```
public class NotifyServiceReceiver extends BroadcastReceiver{

    @Override
    public void onReceive(Context arg0, Intent arg1) {
            …
        }
    }
}
```

  - Register for receiving certain kinds of Intents

```
IntentFilter intentFilter = new IntentFilter();
intentFilter.addAction(ACTION);
registerReceiver(notifyServiceReceiver, intentFilter);
```

Broadcast

- Broadcasts are captured in an app/component if a BroadcastReceiver is registered in AndroidManifest.XML and onReceive() is implemented in the code:

```
<receiver android:name=".MyBroadcastReceiver” android:exported="true">
 <intent-filter>
   <action android:name="android.intent.action.BOOT_COMPLETED"/>
   <action android:name="android.intent.action.INPUT_METHOD_CHANGED”/>
 </intent-filter>
</receiver>
```

```
public class MyBroadcastReceiver extends BroadcastReceiver {
     @Override
     public void onReceive(Context context, Intent intent) {
….
```

BroadcastReceiver

- Receive events announced by other components
- Events announced via Intents
  - Not the same Intent as the one starting an Activity: this one remains in the background
- Events can be announced within your app or publicly to every app on the phone
  - Announce via sendBroadcast()
- Events captured if the receiver is registered:
  - onReceiverRegistered() and then onReceive()

Service, BroadcastReceiver Example

Note on Foreground Services

- Likely to see increased use
  - Google aims to minimize background processing
  - FS for immediate guaranteed tasks, such as mobile payments, apps for unlocking garages, etc.
- In API 26 and above
  - Starting a foreground service should be done with:
  - startForegroundService() – a promise that it will go to foreground and show a notification
    followed by
  - startForeground() – the actual notification is shown

Usually in the Service's onCreate

IntentService

- A Service that
  - Runs on a separate thread
  - Queues up requests and processes them one by one
- Suitable for long running one-off tasks when we don't want to affect the UI responsiveness
- IntentService survives Activity lifecycle changes
- Called using explicit Intent
- Starts on demand, stops when it runs out of work

IntentService

- Define in AndroidManifest.XML

```
<service
    android:name=".FetchAddressIntentService"
    android:exported="false"/>
```

- Extend the class in your Java code

```
public class FetchAddressIntentService extends
IntentService {
```

Invoking IntentService

- Create an explicit Intent for your IntentService
- Use startService() to start the IntentService
- Add additional data if needed with the extra field

Handling Results – from IntentService to Activity (1)

- BroadcastReceiver in your Activity
  - Subclass BroadcastReceiver, implement onReceive
  - Register the receiver for a particular action for times when you would like to handle IntentService results (usually when your Activity is in the foreground)
- Broadcast from your IntentService
  - sendBroadcast() from your IS using the same Intent action as the above

Handling Results – from IntentService to Activity (2)

- ResultReceiver in your Activity
  - Subclass ResultReceiver, implement onReceiveResult

```
class AddressResultReceiver extends ResultReceiver {
    public AddressResultReceiver(Handler handler) {
        super(handler);
    }

    @Override
    protected void onReceiveResult(int resultCode,
                                   Bundle resultData) {…}
```

  - Pass ResultReceiver through Intent when starting IS

```
Intent intent = new Intent(this, FetchAddressIntentService.class);
intent.putExtra(Constants.RECEIVER, mResultReceiver);
intent.putExtra(Constants.LOCATION_DATA_EXTRA, mLastLocation);
startService(intent);
```

Handling Results – from IntentService to Activity (2)

- Set ResultReceiver result
  - IntentService sends results to ResultReceiver in a Bundle with send() method

```
Bundle bundle = new Bundle();
bundle.putString(Constants.RESULT_DATA_KEY, message);
mReceiver.send(resultCode, bundle);
```

- Example
  - Display location address

http://developer.android.com/training/location/display-address.html

IntentService Example

Based on:
https://www.vogella.com/tutorials/AndroidServices/article.html