Chapter 7
Services and Background tasks

Outline

- When and How to use a Service
- Forms of Service
- Service lifecycle
- Creating Started Service
- Creating Bound Service
- AsyncTask
- System Services

When and How to use a Service

- A Service is an application component that can perform long-running operations in the background and does not provide a user interface

- All Android components are running on the main thread of the app process (including Service)
  - Move any long-running operation to a background thread and make sure that this thread is started and controlled by a Service.
  - It is possible to use a background thread from within the Activity instead of moving it to a Service.
  - We use a Service because of the difference in the lifecycle

Forms of Service

- A service can essentially take two forms:
  - **Started**: A service is "started" when an application component (such as an activity) starts it by calling startService( ). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. When the operation is done, the service should stop itself.

  - **Bound:** A service is "bound" when an application component binds to it by calling bindService( ). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, ... A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

- **Caution:** A service runs in the main thread of its hosting process—the service does **not** create its own thread and does **not** run in a separate process (unless you specify otherwise).
  - This means that, if your service is going to do any CPU intensive work or blocking operations (such as MP3 playback or networking), you should create a new thread within the service to do that work

Create a Service

- Create a subclass of **Service**

- Override some callback methods that handle key aspects of the service lifecycle.

- Most important callback methods you should override:
  - onStartCommand(): The system calls this method when another component, such as an activity, requests that the service be started, by calling startService(). If you implement this, it is your responsibility to stop the service when its work is done, by calling stopSelf() or stopService(). (If you only want to provide binding, you don't need to implement this method.)

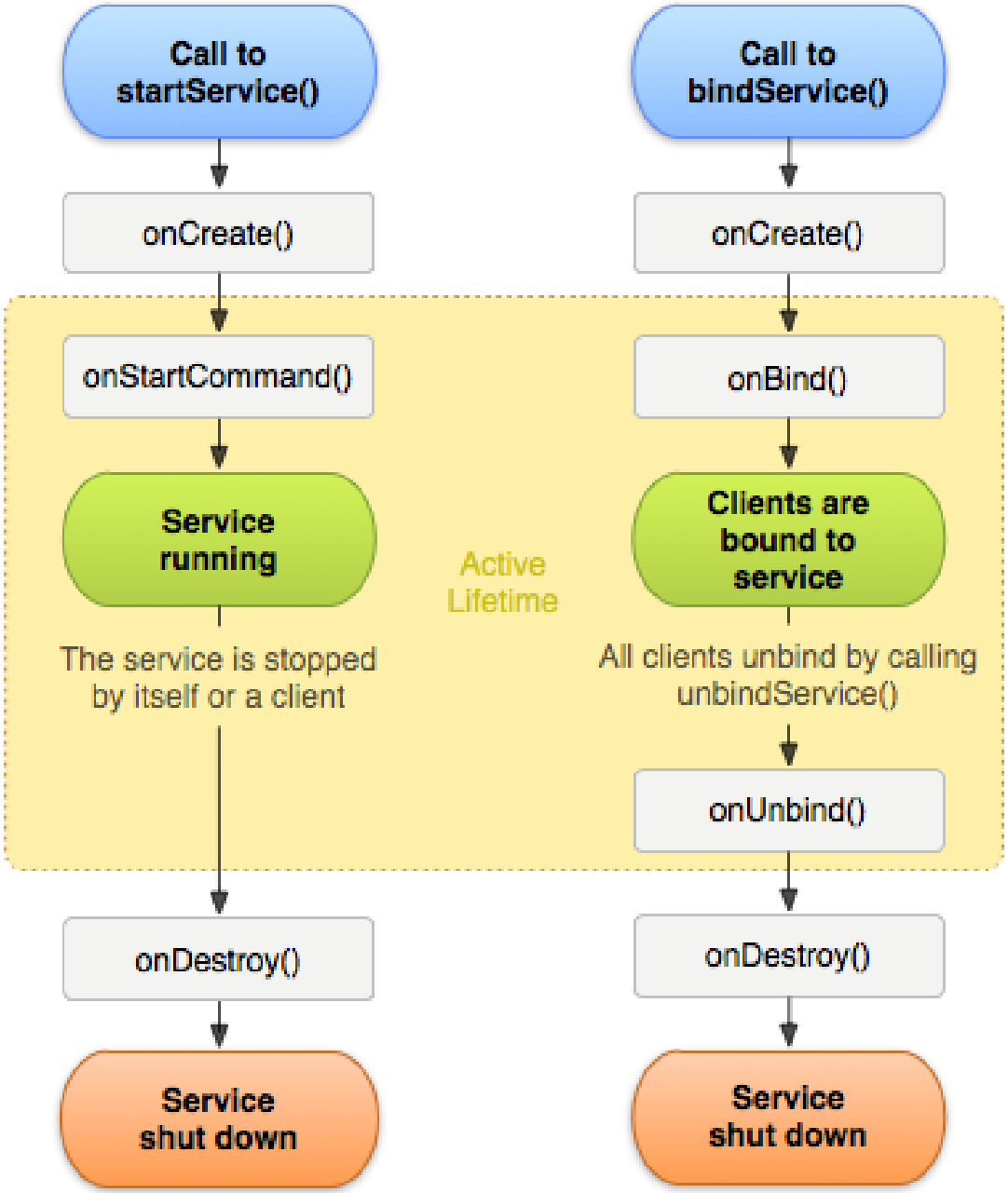
Create a Service

- onBind(): The system calls this method when another component wants to bind with the service, by calling bindService(). In your implementation of this method, you must provide an interface that clients use to communicate with the service, by returning an IBinder. You must always implement this method, but if you don't want to allow binding, then you should return null.
- onCreate(): The system calls this method when the service is first created, to perform one-time setup procedures (before it calls either onStartCommand() or onBind()). If the service is already running, this method is not called.

Create a Service

- onDestroy(): The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc. This is the last call the service receives.

- If a component starts the service by calling startService() (which results in a call to onStartCommand()), then the service remains running until it stops itself with stopSelf() or another component stops it by calling stopService().
- If a component calls bindService() to create the service (and onStartCommand() is *not* called), then the service runs only as long as the component is bound to it. Once the service is unbound from all clients, the system destroys it.


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Active Lifetime
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service

Declaring a service in the manifest

- Like activities, you must declare all services in your application's manifest file

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Creating a Started Service - Example

```
public class HelloService extends Service {
@Override
public void onCreate() {
Toast.makeText(this, "service created ", Toast.LENGTH_SHORT).show();
Runnable r = new Runnable() {
   public void run() {
   // Normally we would do some work here, like download a file. For our sample, we just sleep for 15 seconds.
        long endTime = System.currentTimeMillis() + 15*1000;
        while (System.currentTimeMillis() < endTime) {
                try {
                     wait(endTime - System.currentTimeMillis());
                }
                catch (Exception e) {
                }
        }
        stopSelf();
    }
 };
Thread t = new Thread(r);
t.start();
}
}
```

Creating a Started Service - Example

```
// Still in the HelloService class
@Override
public int onStartCommand(Intent intent, int flags, int startId)
{
        Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
 return START_STICKY; // If we get killed (example out of memory), after returning from here, restart

// More return options: START_NOT_STICKY, START_REDELIVER_INTENT
}

@Override
public IBinder onBind(Intent arg0) {
        return null;
}

@Override
public void onDestroy()
{
        Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();
}
}// End of HelloService
```

Creating a Started Service - Example

- Add this <service android:name=".HelloService"></service> to the manifest file

- To start a Service from an Activity

```
public void startHelloService(View view)
{
Intent intent = new Intent(this, HelloService.class);
startService(intent);
}
```

- To stop explicitly a service

```
public void stopHelloService(View view)
{
        stopService(new Intent(this, HelloService.class));
}
```

Application: File download

- Show **DownloadService** App
  - Downloading a distant file to external storage
  - Displaying Progress in a Notification

Bound Service

- While working with Android services, there comes a situation where we would want the service to communicate with an activity

- To accomplish this task one has to bind a service to an activity, this type of service is called an android **bound service**.

- A bound service is one that allows application components to bind to it by calling bindService()

- To create a bound service, you must implement the onBind() callback method to return an Ibinder that defines the interface for communication with the service

- The service lives only to serve the application component that is bound to it, so when there are no components bound to the service, the system destroys it

- Multiple clients can bind to the service at once. When a client is done interacting with the service, it calls **unbindService()** to unbind

Create a bound service class

```
public class BoundService extends Service {
private IBinder binder = new MyBinder();
@Override
public void onCreate()
{ … }
@Override //Return the communication channel to the service
public IBinder onBind(Intent arg0) {
        return binder;
}
@Override // Called when new clients have connected to the service, after it had previously been notified that all had disconnected in its onUnbind(Intent)
public void onRebind(Intent intent) {
        super.onRebind(intent);
}
```

Create a bound service class

```
@Override
public boolean onUnbind(Intent intent)
{
        return true; // on next bind call onRebind only.
}

public class MyBinder extends Binder{

BoundService getService()
{

            return BoundService.this;
}

}

} // end of BoundService class
```

Create a main activity

```
public class MainActivity extends Activity {
private BoundService boundService;
private boolean isBound = false;
@Override
protected void onCreate(Bundle savedInstanceState) {
            ...
}
@Override
public void onStart()
{
            super.onStart();

            Intent intent = new Intent(this, BoundService.class);
            startService(intent); // why we need to start the service ?
            bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE);
}
```

Create a main activity

```
@Override
public void onStop()
{
        super.onStop();
        if(isBound)
        {
                unbindService(serviceConnection);
                isBound = false;
        }
}
```

Create a main activity

```
private ServiceConnection serviceConnection = new ServiceConnection(){
@Override
public void onServiceConnected(ComponentName name, IBinder service) {
MyBinder binder = (MyBinder)service;
boundService = binder.getService();
isBound = true;
}
@Override
public void onServiceDisconnected(ComponentName name) {
            isBound = false;
}
};
} // end of MainActivity
```

Background operations with AsyncTask

- AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread

- AsyncTasks should ideally be used for short operations (a few seconds at the most)

- An asynchronous task is defined by 3 generic types, called Params, Progress and Result
  - android.os.AsyncTask<Params, Progress, Result>

- 4 steps: called onPreExecute, doInBackground, onProgressUpdate and onPostExecute

Background operations with AsyncTask

- onPreExecute(), invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
- doInBackground(Params...), invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time.

- onProgressUpdate(Progress...), invoked on the UI thread after a call to publishProgress(Progress...). This method is used to display any form of progress in the user interface while the background computation is still executing.

- onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

Background operations with AsyncTask

- So:
  - The AsyncTask executes everything in **doInBackground**() inside of another thread, which does not have access to the GUI where your views are.
  - **preExecute**(), **postExecute**(), and **onProgressUpdate**(Progress) offer you access to GUI

AsyncTask - Example

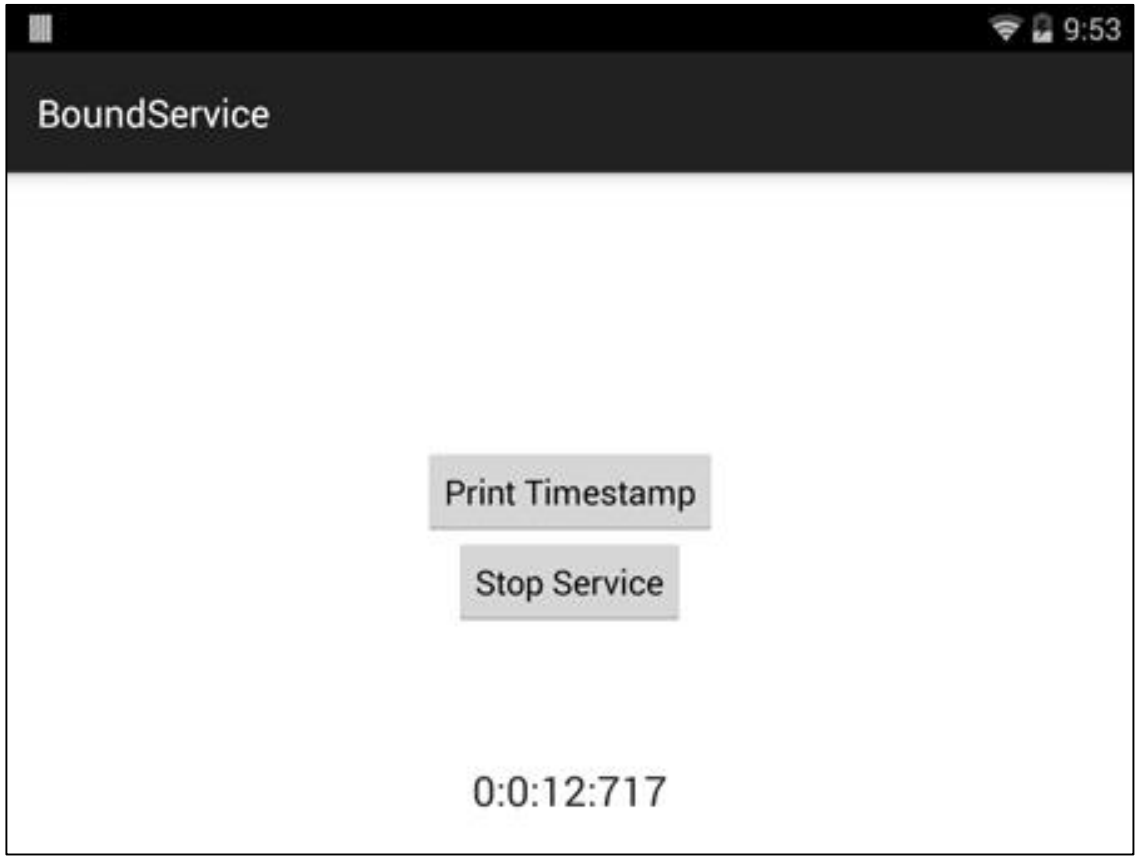
- Show AsyncTaskDemo
  - Long time operation
  - Show the progress in a ProgressBar

Lab 1

- Modify the AsyncTaskDemo
  - Integrate the Download service
  - ProgressBar should display the download percentage (in addition to the notification)
  - Allow the user to control the download (stop, pause, resume)

Lab 2

- Create an app that allows to start a timer service which would keep counting a timer from the start of service



System Services

- **Object android.app.Activity.getSystemService(String name)**
- Return the handle to a system-level service by name. The class of the returned object varies by the requested name. Currently available names are:
  - POWER_SERVICE ("power"): A PowerManager for controlling power management.
  - ALARM_SERVICE ("alarm"): A AlarmManager for receiving intents at the time of your choosing.
  - NOTIFICATION_SERVICE ("notification"): A NotificationManager for informing the user of background events.
  - LOCATION_SERVICE ("location"): A LocationManager for controlling location (e.g., GPS) updates.
  - SEARCH_SERVICE ("search"): A SearchManager for handling search.
  - VIBRATOR_SERVICE ("vibrator"): A Vibrator for interacting with the vibrator hardware.
  - WIFI_SERVICE ("wifi"): A WifiManager for management of Wi-Fi connectivity.
  - DOWNLOAD_SERVICE ("download"): A DownloadManager for requesting HTTP downloads
  - BATTERY_SERVICE ("batterymanager"): A BatteryManager for managing battery state
  - …

Notification Service

```
NotificationManager nm =
(NotificationManager)getSystemService(NOTIFICATION_SERVICE);

…

nm.notify(notificationID, notif);
```

Download Service - Example

```
String link = editText.getText().toString();

DownloadManager downloadManager =
            (DownloadManager)getSystemService(DOWNLOAD_SERVICE);

Uri uri = Uri.parse(link);

DownloadManager.Request request = new DownloadManager.Request(uri);
request.setAllowedNetworkTypes(DownloadManager.Request.NETWORK_MOBILE |
                               DownloadManager.Request.NETWORK_WIFI);
request.setTitle("Downloading file");
request.setDescription("Download in progress");

String [] temp = uri.getPath().split("/");
String fileName = temp[temp.length - 1];

request.setDestinationInExternalFilesDir(getApplicationContext(),
                               Environment.DIRECTORY_DOWNLOADS, fileName);

downloadManager.enqueue(request);
```

Battery Service

```
public void batteryLevel(View view)
{
BatteryManager bm = (BatteryManager)getSystemService(BATTERY_SERVICE);

String level = "Battery capacity " + bm.BATTERY_PROPERTY_CAPACITY;
Toast.makeText(getApplicationContext(), level, Toast.LENGTH_SHORT).show();
}

public void powerPlugedUSB(View view)
{
BatteryManager bm = (BatteryManager)getSystemService(BATTERY_SERVICE);

String amp = "battery current in microamperes : " +
bm.getIntProperty(BatteryManager.BATTERY_PROPERTY_CURRENT_NOW);
Toast.makeText(getApplicationContext(), amp, Toast.LENGTH_SHORT).show();
}
```

Location Service

```
public void showLocationProviders(View view) {
LocationManager lm = (LocationManager)getSystemService(LOCATION_SERVICE);
List<String> providers = lm.getProviders(true);

String listProviders = "";
if(providers.size() != 0){
        Iterator<String> it = providers.iterator();
        while(it.hasNext())
                listProviders += it.next() + " - ";
}
else
        listProviders = "No location providers enabled";

Toast.makeText(getApplicationContext(), listProviders, Toast.LENGTH_LONG).show();
}
```

passive
network
gps

Location Service

```
public void showMyLocation(View view){
LocationManager lm = (LocationManager)getSystemService(LOCATION_SERVICE);
List<String> providers = lm.getProviders(true);

if(providers.size() != 0) {
        String provider = providers.get(0);
        Location l = lm.getLastKnownLocation(provider);

        Toast.makeText(getApplicationContext(), l.getLatitude() + "," + l.getLongitude(),
        Toast.LENGTH_LONG).show();
        }
}
```

Network provider needs

android.permission.ACCESS_COARSE_LOCATION

or

android.permission.ACCESS_FINE_LOCATION

GPS and passive providers need *android.permission.ACCESS_FINE_LOCATION*

Location Service

Use the following method from LocationManager class to register for location updates using the named provider

public void **requestLocationUpdates** (String provider, long minTime, float minDistance, LocationListener listener)

provider: the name of the provider with which to register
minTime: minimum time interval between location updates, in milliseconds
minDistance: minimum distance between location updates, in meters
listener: a LocationListener whose onLocationChanged(Location) method will be called for each location update

Google Play services location APIs

- Getting the Last Known Location
  - It use the fused location provider to retrieve the device's last known location
  - Fused Location Provider analyses GPS, Cellular and Wi-Fi network location data in order to provide the highest accuracy data
  - Fused location is one of the location APIs in Google Play services
- Main classes used
  - LocationServices.FusedLocationApi
  - GoogleApiClient.Builder
- Show LocationUpdates app