Mobile Application Development

Higher Diploma in Science in Computer Science

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

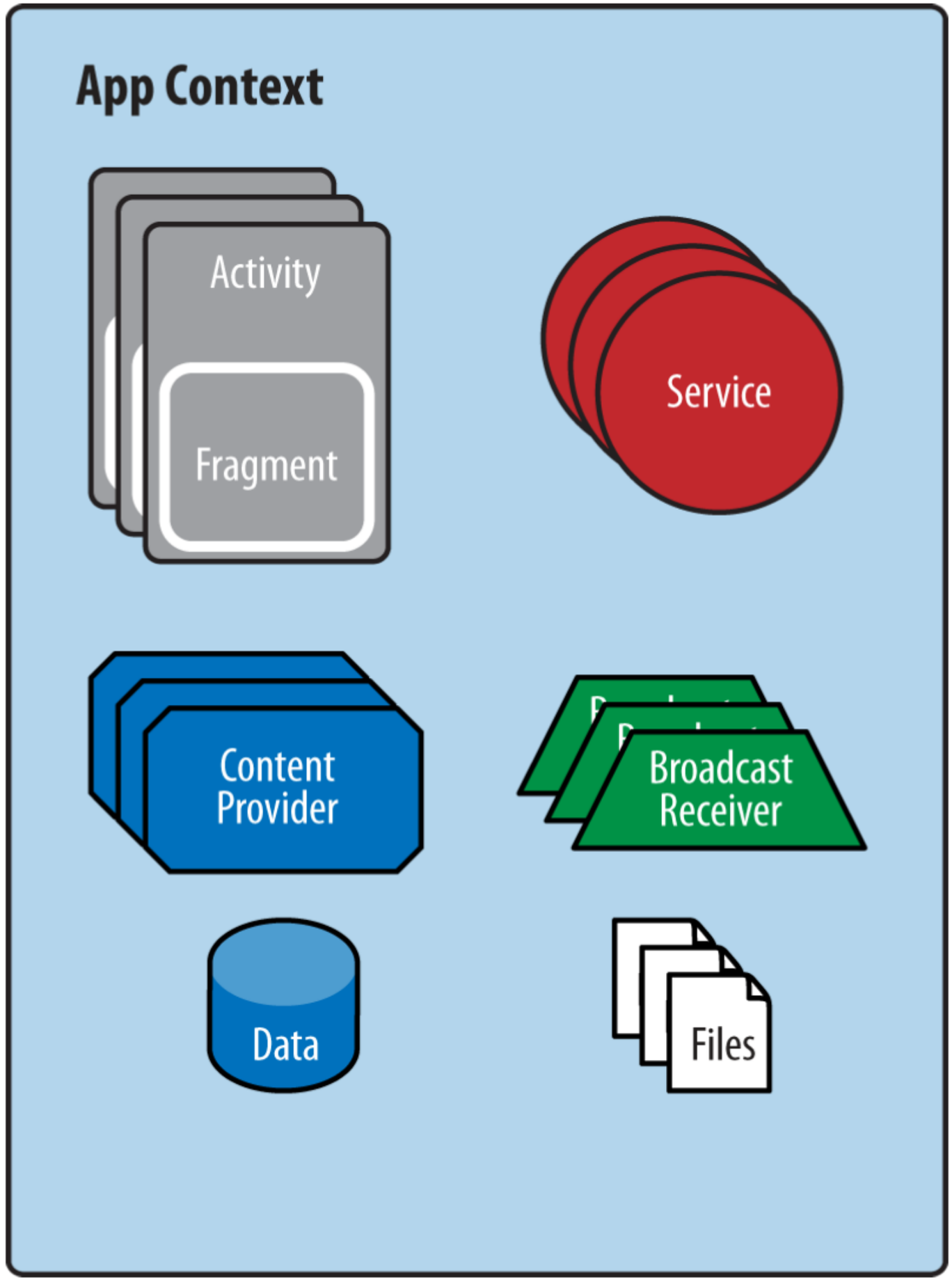
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Services & BroadcastReceivers

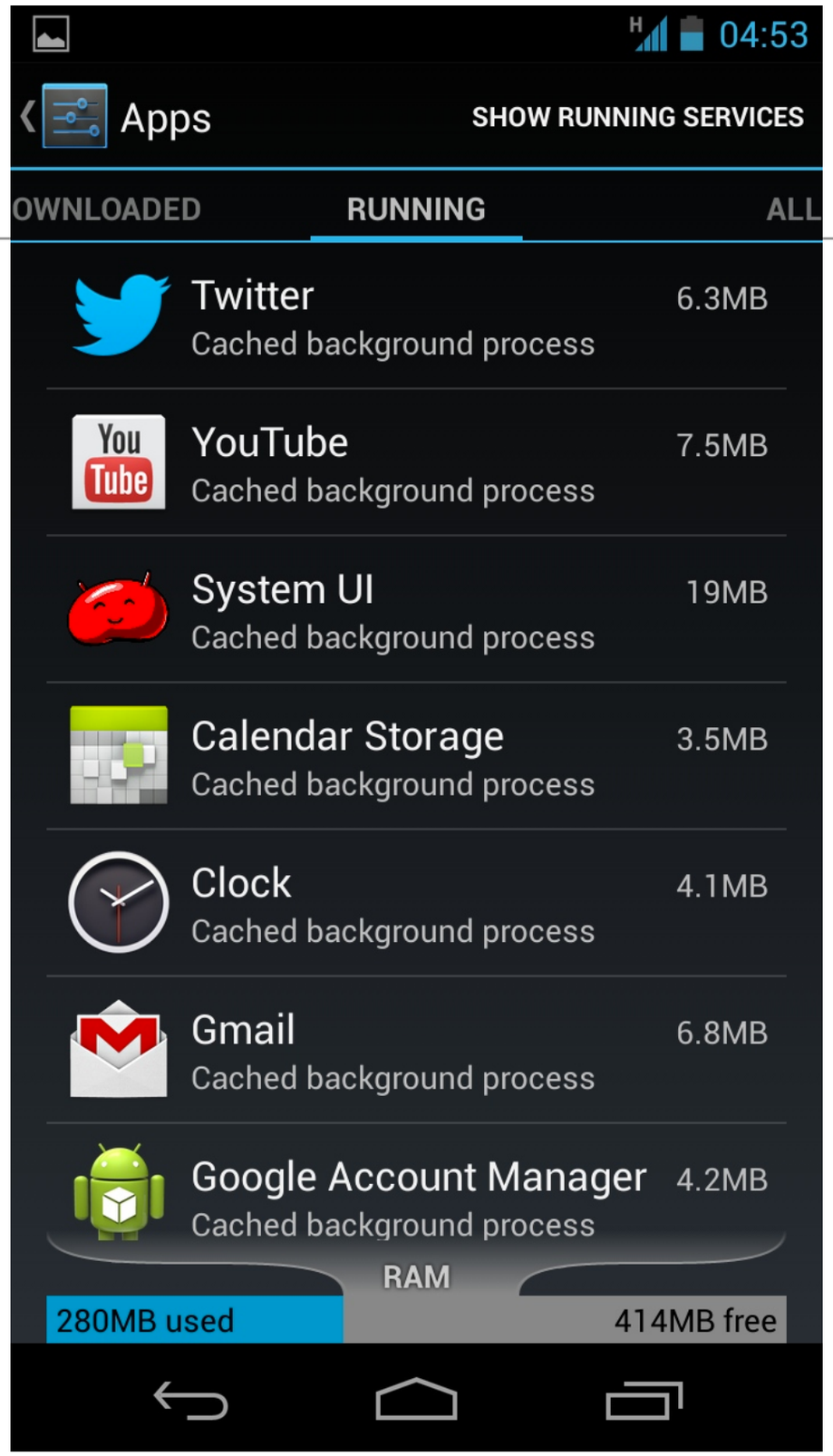
Application Context

- Services
  - IntentServices
  - System Services
  - Alarms
- Broadcast Receivers
  - BootReceivers



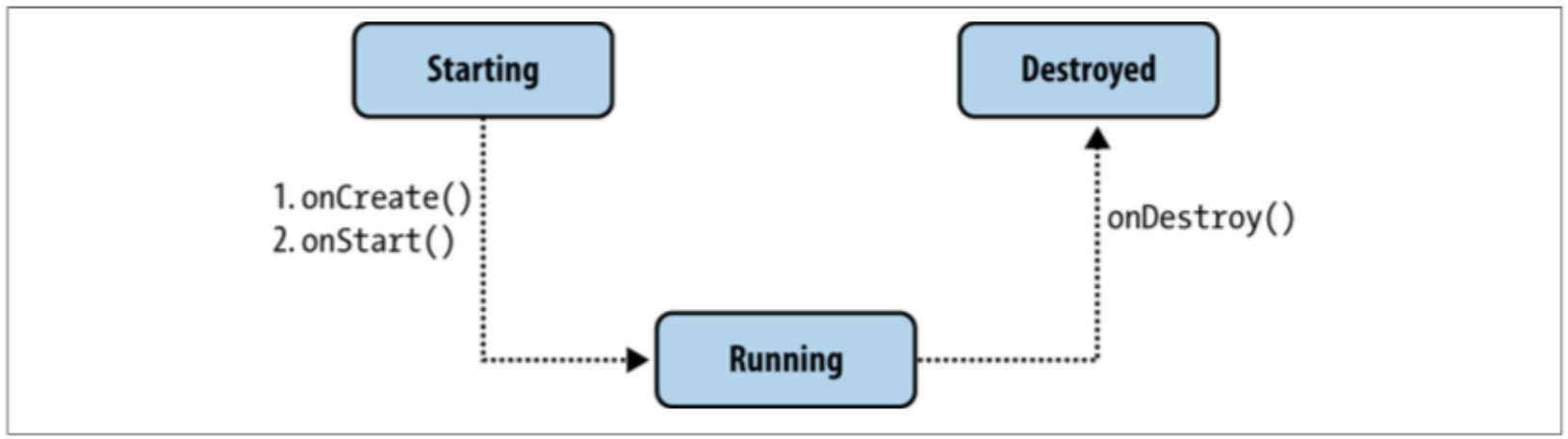
Services

- Services are among the main building blocks in Android.
- Unlike an activity, a service doesn't have a user interface; it is simply a piece of code that runs in the background of your application.
- Services are used for processes that should run independently of activities, which may come and go.



Service Lifecycle

- Just like an activity, a service has a well-defined life cycle



- Create the Java class representing the service.
- Register the service in the AndroidManifest.xml file.
- Start the service

- onBind() is used in bound services to return the actual implementation of something called a binder. Bound services can provide more specific APIs to other applications via an interface

- onCreate() is called when the service is initially created. It is not called for subsequent startService() calls

- onStartCommand() is called each time the service receives a startService() intent. A service that is already started could get multiple requests to start again, and each will cause onStartCommand() to execute. START_STICKY is used as a flag to indicate this service is started and stopped explicitly

- onDestroy() is called just before the service is destroyed by the stopService() request.

```
package app.services;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class BackgroundService extends Service
{
  static final String TAG = "RefreshService";

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

  @Override
  public void onCreate()
  {
    super.onCreate();
    Log.d(TAG, "onCreated");
  }

  @Override
  public int onStartCommand(Intent intent, int flags, int startId)
  {
    super.onStartCommand(intent, flags, startId);
    Log.d(TAG, "onStarted");
    return START_STICKY;
  }

  @Override
  public void onDestroy()
  {
    super.onDestroy(); Log.d(TAG, "onDestroyed");
  }
}
```

Intent Services

- The IntentService class provides a straightforward structure for running an operation on a single background thread.

- This allows it to handle long-running operations without affecting your user interface's responsiveness.

- An IntentService isn't affected by most user interface lifecycle events

IntentService Class

- Standard Services run on the main thread of the application, i.e., the UI thread.
- This implies that the service is unsuitable for making network calls.
- An intent service is similar to regular service, with two main exceptions: whatever work is to be done in onHandleIntent() will execute on a separate worker thread, and once it's done, the service will stop.

public abstract class

IntentService

extends Service

java.lang.Object
↳android.content.Context
↳android.content.ContextWrapper
↳android.app.Service
↳android.app.IntentService

Class Overview

IntentService is a base class for `Service` s that handle asyn
the service is started as needed, handles each Intent in turn

This "work queue processor" pattern is commonly used to of
mechanics. To use it, extend IntentService and implement `o`
appropriate.

All requests are handled on a single worker thread -- they ma
time.

Developer Guides

Intent Service Limitations

- It can't interact directly with your user interface. To put its results in the UI, you have to somehow send them to an Activity.

- Work requests run sequentially. If an operation is running in an IntentService, and you send it another request, the request waits until the first operation is finished.

- An operation running on an IntentService can't be interrupted.

- In most cases an IntentService is the preferred way to simple background operations.

Creating an IntentService

- To create an IntentService define a class that extends IntentService, and within it, define a method that overrides onHandleIntent().

```
public class RSSPullService extends IntentService {
    @Override
    protected void onHandleIntent(Intent workIntent) {
        // Gets data from the incoming Intent
        String dataString = workIntent.getDataString();
        ...
        // Do work here, based on the contents of dataString
        ...
    }
}
```

Define the IntentService in the Manifest

- An IntentService also needs an entry in your application manifest.
- Provide this entry as a <service> element that's a child of the <application> element:

```
<application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
    ...
    <!--
        Because android:exported is set to "false",
        the service is only available to this app.
    -->
    <service
        android:name=".RSSPullService"
        android:exported="false"/>
    ...
<application/>
```

- The attribute android:name specifies the class name of the IntentService.

Creating an IntentService Request

- To create a work request and send it to an IntentService, create an explicit Intent, add work request data to it, and send it to IntentService by calling startService().

```
/*
 * Creates a new Intent to start the RSSPullService
 * IntentService. Passes a URI in the
 * Intent's "data" field.
 */
mServiceIntent = new Intent(getActivity(), RSSPullService.class);
mServiceIntent.setData(Uri.parse(dataUrl));
```

Starting the IntentSerivce

```
// Starts the IntentService
getActivity().startService(mServiceIntent);
```

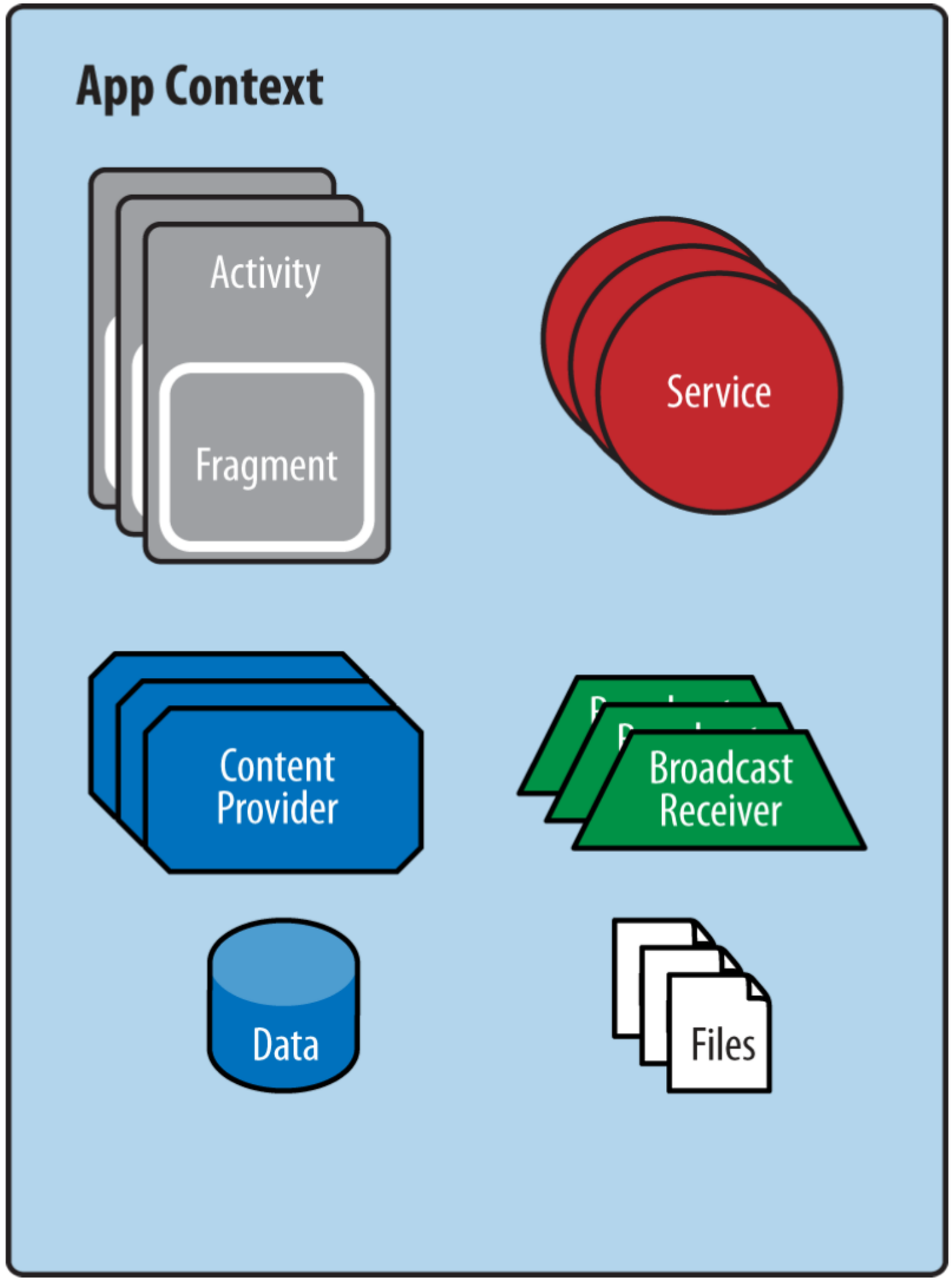
- Notice that you can send the work request from anywhere in an Activity or Fragment. For example, if you need to get user input first, you can send the request from a callback that responds to a button click or similar gesture.

- Once you call startService(), the IntentService does the work defined in its onHandleIntent() method, and then stops itself.

Broadcast Receivers

- Broadcast receivers are Android's implementation of a system-wide publish/ subscribe mechanism.

- The receiver is dormant code that gets activated by the occurrence of an event to which the receiver is subscribed. The "event" takes the form of an intent.



Broadcast Receivers

- The system itself broadcasts events all the time. For example
  - when an SMS arrives
  - a call comes in
  - the battery runs low
  - the system completes booting up
- All those events are broadcast, and any number of receivers could be triggered by them.

Boot Receiver

- A broadcast receiver that the system will launch when the boot is complete

- Must be registered in the Manifest

```
package app.services;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

public class BootReceiver extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
    Log.d("BootReceiver", "onReceived");
  }
}
```

```
<receiver android:name=".BootReceiver">

  <intent-filter>

     <action android:name="android.intent.action.BOOT_COMPLETED" />

  </intent-filter>

</receiver>
```

Broadcasting Intents

- We can broadcast an intent from anywhere in our application

- If any components - Activities for instance - are interested in the event/intent - they can register to receive the event.

- This is carried out via a ‘LocalBroadCastManager’ object

Report Status From an IntentService

- To send the status of a work request in an IntentService to other components, first create an Intent that contains the status in its extended data

```
public final class Constants {
    ...
    // Defines a custom Intent action
    public static final String BROADCAST_ACTION =
        "com.example.android.threadsample.BROADCAST";
    ...
    // Defines the key for the status "extra" in an Intent
    public static final String EXTENDED_DATA_STATUS =
        "com.example.android.threadsample.STATUS";
    ...
}
```

- The send the Intent by calling LocalBroadcastManager.sendBroadcast().

- This sends the Intent to any component registered to receive it.

```
public class RSSPullService extends IntentService {
...
    /*
     * Creates a new Intent containing a Uri object
     * BROADCAST_ACTION is a custom Intent action
     */
    Intent localIntent =
            new Intent(Constants.BROADCAST_ACTION)
            // Puts the status into the Intent
            .putExtra(Constants.EXTENDED_DATA_STATUS, status);
    // Broadcasts the Intent to receivers in this app.
    LocalBroadcastManager.getInstance(this).sendBroadcast(localIntent)
...
}
```

Receive Status Broadcasts from an IntentService

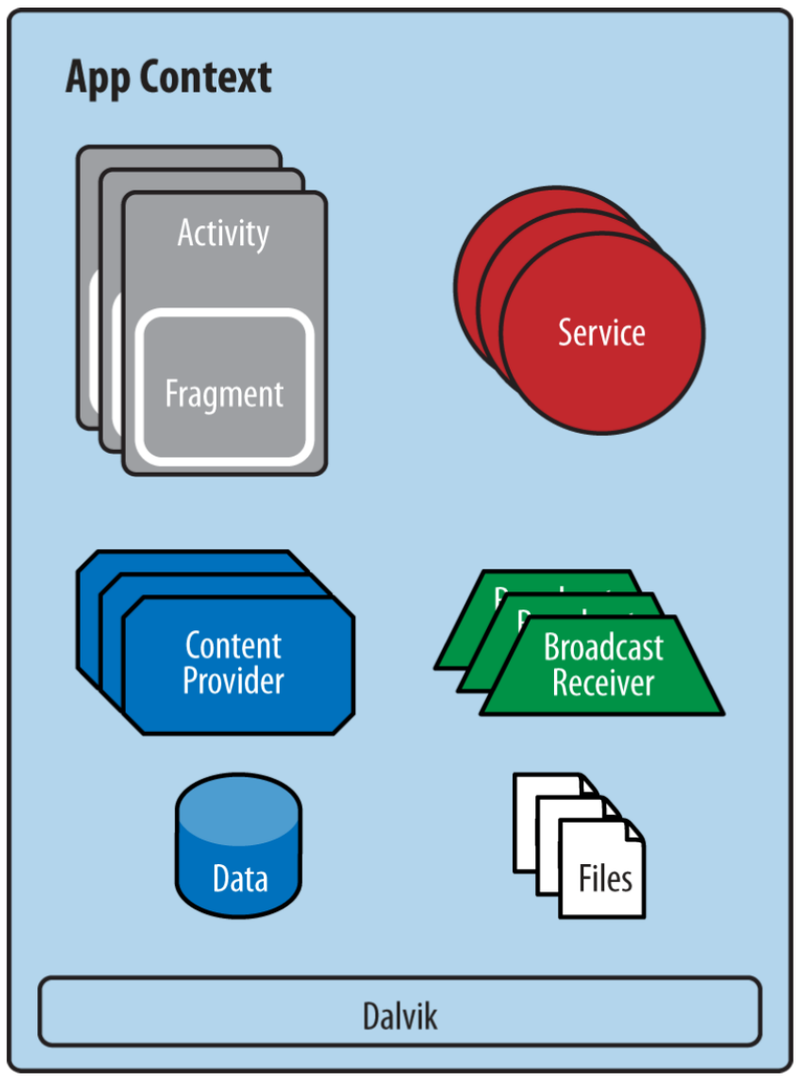
To receive broadcast Intent objects, use a subclass of BroadcastReceiver.

In the subclass, implement the BroadcastReceiver.onReceive() callback method, which LocalBroadcast Manager invokes when it receives an Intent..

```
// Broadcast receiver for receiving status updates from the IntentService
private class ResponseReceiver extends BroadcastReceiver
{
    // Prevents instantiation
    private DownloadStateReceiver() {
    }
    // Called when the BroadcastReceiver gets an Intent it's registered to receive
    @
    public void onReceive(Context context, Intent intent) {
...
        /*
         * Handle Intents here.
         */
...
    }
}
```

Systems Services

- System services - always-on-always-running processes.
- There are around 60+ of these services, such as
- The Service API is well documented for each service.
- What is common for all of them is that they are readily available to your app via the context



- Alarm
- Audio
- Camera,
- Media
- Location
- Sensors
- Telephony
- USB
- WiFi,

Scheduling Repeating Alarms

- Alarms (based on the AlarmManager class) give you a way to perform time-based operations outside the lifetime of your application.
- E.g use an alarm to initiate a long-running operation, such as starting a service once a day to download a weather forecast.

Alarm Characteristics

- 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

The Alarm Service - Example

```
public class BootReceiver extends BroadcastReceiver
{
  public static int REQUESTCODE = -1;
  private static final long DEFAULT_INTERVAL = AlarmManager.INTERVAL_FIFTEEN_MINUTES;

  @Override
  public void onReceive(Context context, Intent intent)
  {
    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
    long interval = DEFAULT_INTERVAL;

    PendingIntent operation = PendingIntent.getService(context,
                                                       REQUESTCODE,
                                                       new Intent(context, RefreshService.class),
                                                       PendingIntent.FLAG_UPDATE_CURRENT);

    AlarmManager alarmManager = (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);

    alarmManager.setInexactRepeating(AlarmManager.RTC, System.currentTimeMillis(), interval, operation);
  }
}
```

- A “PendingIntent”, as created above, is the intent to be generated by the AlarmService at the requested interval

- The intent ‘RefreshService’ will be sent to the host application

RefreshReceiver

- When we get the intent, fetch the latest donations from the donation service
- Broadcast the receipt of those donations to interested parties

```
public class RefreshService extends IntentService
{
  DonationApp app;

  public RefreshService()
  {
    super("RefreshService");
  }

  @Override
  public void onCreate()
  {
    super.onCreate();
    LogHelpers.info(this, "onCreated");
    app = (DonationApp)getApplication();
  }

  @Override
  protected void onHandleIntent(Intent intent)
  {
    app = (DonationApp) getApplication();
    Intent localIntent = new Intent(Report.BROADCAST_ACTION);
    Call<List<Donation>> call = (Call<List<Donation>>) app.donationService.getAllDonations();
    try
    {
      Response<List<Donation>> response = call.execute();
      app.donations = response.body();
      LocalBroadcastManager.getInstance(this).sendBroadcast(localIntent);
    }
    catch (IOException e)
    {
      LogHelpers.info(tag, "Failed to retrieve donations list - network error");
    }
  }
}
```

BroadCastReceiver

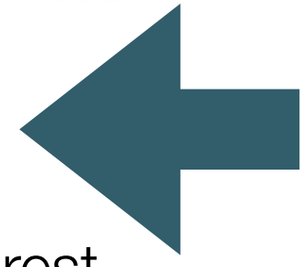
- Register to receive the Broadcast

- When this is received, update the list of donations

```
public class Report extends Activity
{
  public static final String BROADCAST_ACTION = “app.activities.Report";
  //…

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    //…
    intentFilter = new IntentFilter(BROADCAST_ACTION);
    ResponseReceiver mResponseReceiver = new ResponseReceiver();
    LocalBroadcastManager.getInstance(this).registerReceiver(mResponseReceiver, intentFilter);
  }

  private class ResponseReceiver extends BroadcastReceiver
  {
    private void ResponseReceiver()
    {
    }

    @Override
    public void onReceive(Context context, Intent intent)
    {
      adapter.donations = app.donations;
      adapter.notifyDataSetChanged();
    }
  }
 }

//…
```

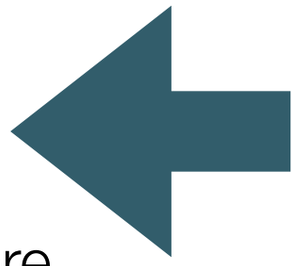
Synchronous VS Asynchronous Retrofit Calls

- Synchronous:
  - ‘Execute’ the call to the remote service
  - Wait until response send back
  - Read response and proceed with rest of program



Do this in unit tests & in Background Android Intent Services

- Asynchronous:
  - ‘Enqueue’ the call to the remote service
  - Provide a ‘callback’ objects to be invoked when response arrives
  - Proceed with rest of program before the response is received



Do this in Activities and other Android UI code

Asynchronous Retrofit Calls

```
public class Report extends AppCompatActivity implements Callback<List<Donation>>
{
 ...

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    ...
    app = (DonationApp) getApplication();
    Call<List<Donation>> call = (Call<List<Donation>>) app.donationService.getAllDonations();
    call.enqueue(this);
  }


  @Override
  public void onResponse(Response<List<Donation>> response, Retrofit retrofit)
  {
    adapter.donations = response.body();
    adapter.notifyDataSetChanged();
  }

  @Override
  public void onFailure(Throwable t)
  {
    Toast toast = Toast.makeText(this, "Error retrieving donations", Toast.LENGTH_LONG);
    toast.show();
  }

}
```

Synchronous Retrofit Calls

- Refresh Service

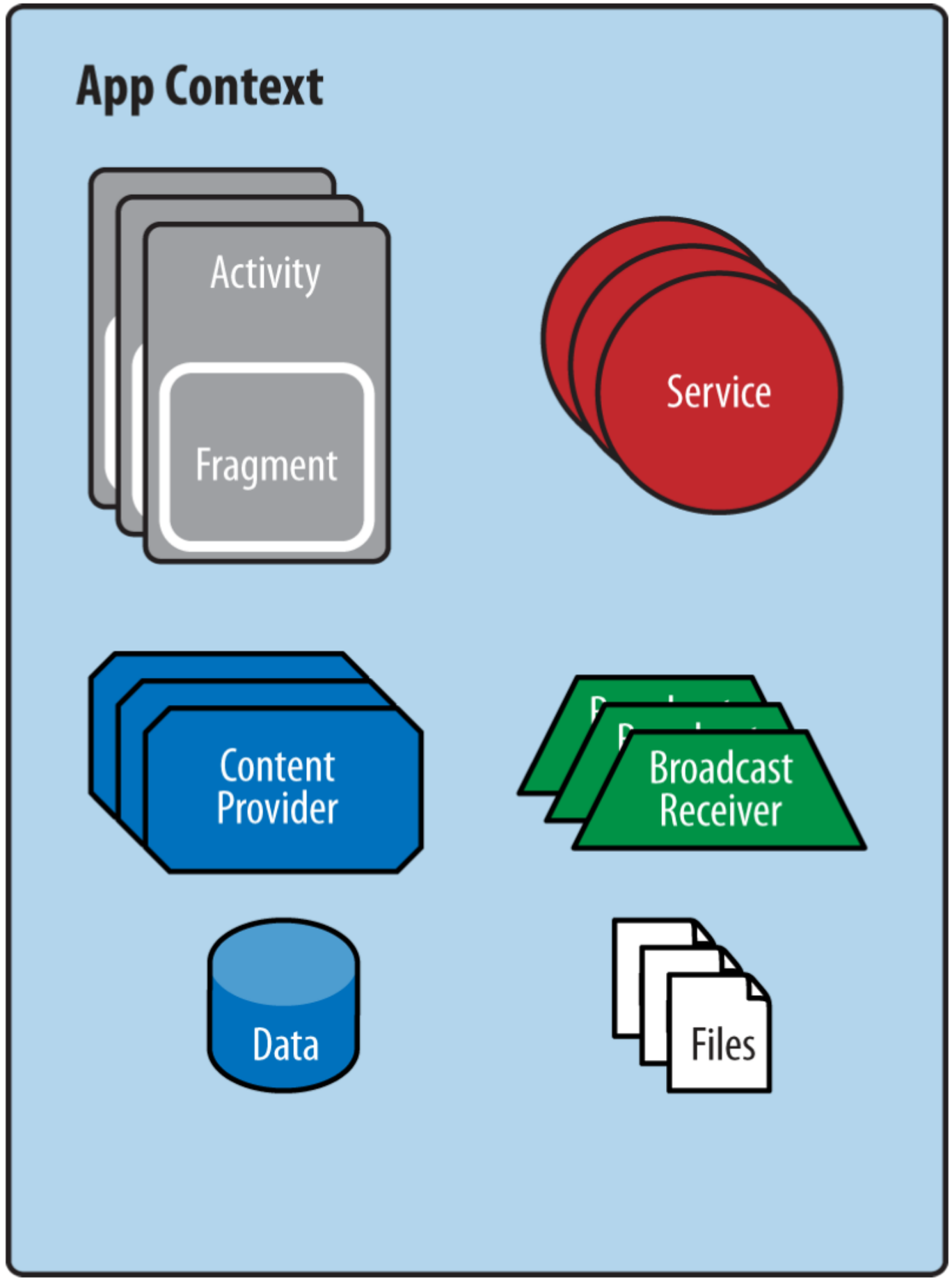
```
public class RefreshService extends IntentService
{
…
  @Override
  protected void onHandleIntent(Intent intent)
  {
    app = (DonationApp) getApplication();
    Intent localIntent = new Intent(Report.BROADCAST_ACTION);
    Call<List<Donation>> call = (Call<List<Donation>>) app.donationService.getAllDonations();
    try
    {
      Response<List<Donation>> response = call.execute();
      app.donations = response.body();
      LocalBroadcastManager.getInstance(this).sendBroadcast(localIntent);
    }
    catch (IOException e)
    {
      LogHelpers.info(tag, "Failed to retrieve donations list - network error");
    }
  }
}
```

- Unit Test

```
public List<Donor> getAllDonors() throws Exception
{
  Call<List<Donor>> call = (Call<List<Donor>>) service.getAllDonors();

  Response<List<Donor>> donors = call.execute();

  return donors.body();
}
```

Application Context

- Services
  - IntentServices
  - System Services
  - Alarms
- Broadcast Receivers
  - BootReceivers



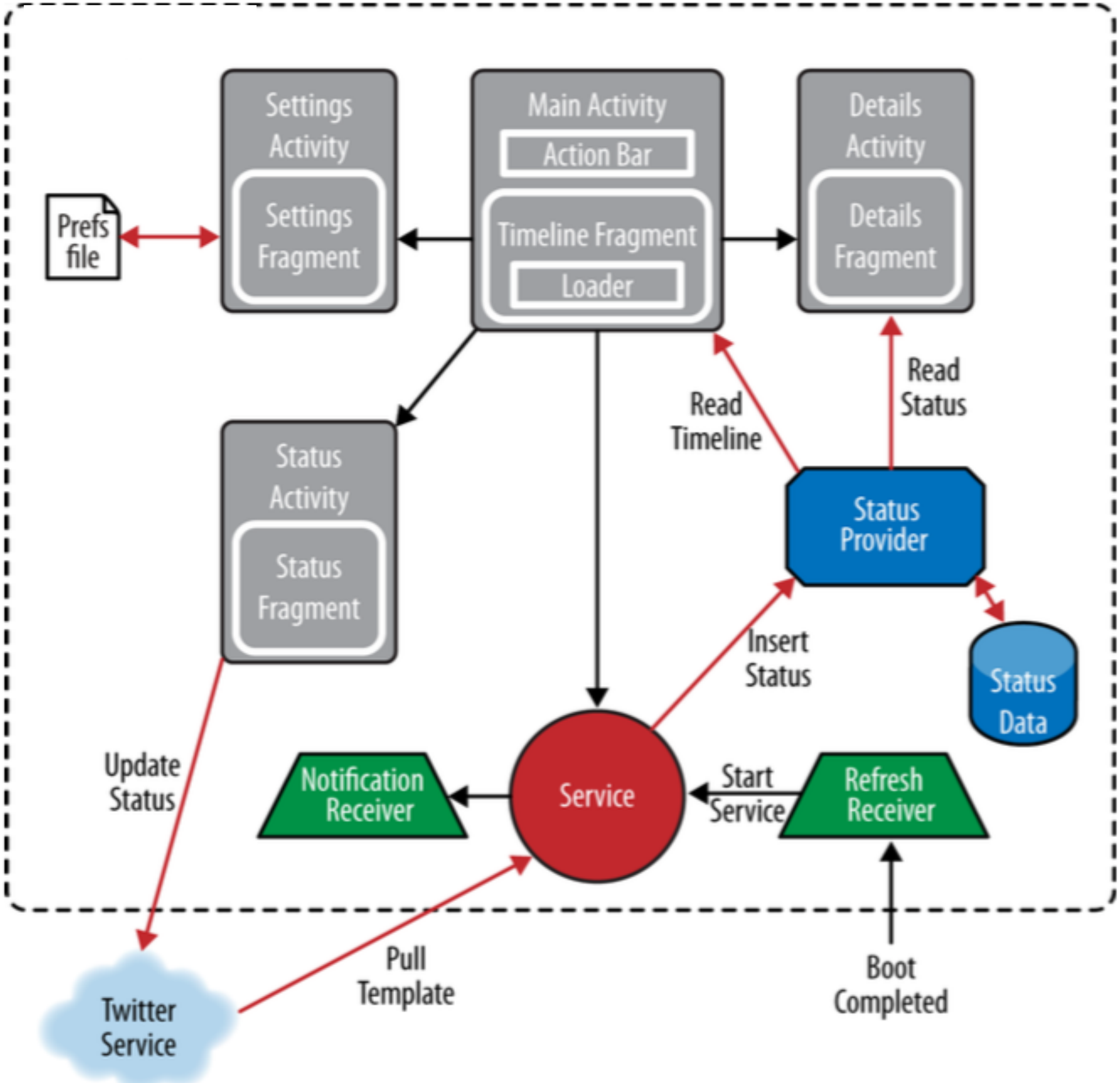
O'REILLY®

2nd Edition

Learning Android

DEVELOP MOBILE APPS USING JAVA AND ECLIPSE

Marko Gargenta &
Masumi Nakamura


Settings Activity
Settings Fragment
Main Activity
Action Bar
Timeline Fragment
Loader
Details Activity
Details Fragment
Prefs file
Status Activity
Status Fragment
Read Timeline
Read Status
Status Provider
Insert Status
Status Data
Update Status
Notification Receiver
Service
Start Service
Refresh Receiver
Twitter Service
Pull Template
Boot Completed