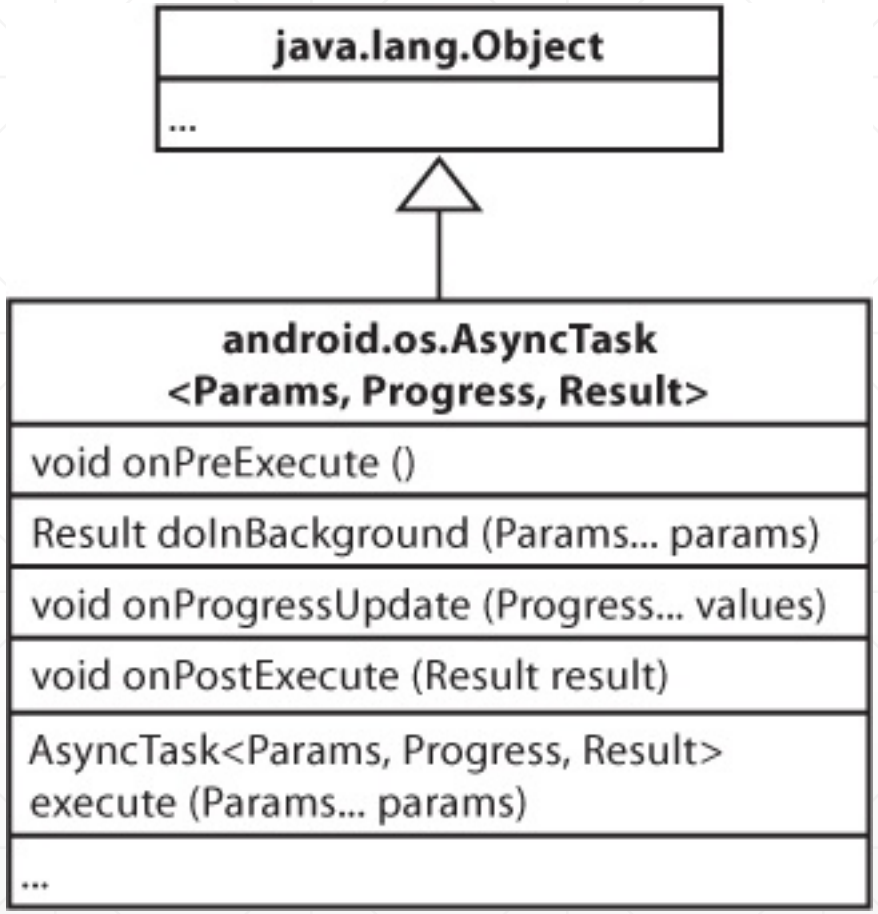
Chapter 12 – Async Threads

- Main event
- Render
- All other threads

```
//Update the database when the checkbox is clicked
public void onFavoriteClicked(View view){

    int drinkNo = (Integer)getIntent().getExtras().get(EXTRA_DRINKNO);
1   CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("FAVORITE", favorite.isChecked());

    SQLiteOpenHelper starbuzzDatabaseHelper =
                              new StarbuzzDatabaseHelper(DrinkActivity.this);
    try {
2       SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
        db.update("DRINK", drinkValues,
                              "_id = ?", new String[] {Integer.toString(drinkNo)});
        db.close();
    } catch(SQLiteException e) {

3       Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
      }
}
```


java.lang.Object
...
android.os.AsyncTask
<Params, Progress, Result>
void onPreExecute ()
Result doInBackground (Params... params)
void onProgressUpdate (Progress... values)
void onPostExecute (Result result)
AsyncTask<Params, Progress, Result>
execute (Params... params)
...

```
private class MyAsyncTask extends AsyncTask<Params, Progress, Result>

        protected void onPreExecute() {
                //Code to run before executing the task
         }

        protected Result doInBackground(Params... params) {
                //Code that you want to run in a background thread
        }

        protected void onProgressUpdate(Progress... values) {
                //Code that you want to run to publish the progress of your task
        }

        protected void onPostExecute(Result result) {
                //Code that you want to run when the task is complete
        }
}
```


```
private class UpdateDrinkTask extends AsyncTask<Params, Progress, Result> {

    ContentValues drinkValues;

    protected void onPreExecute() {
        CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
        drinkValues = new ContentValues();
        drinkValues.put("FAVORITE", favorite.isChecked());
    }

    ...

}
```

Before we run the database code, we need to get the value of the favorite checkbox.

```
private class UpdateDrinkTask extends AsyncTask<Integer, Progress, Boolean> {

    ContentValues drinkValues;

    ...

    protected Boolean doInBackground(Integer... drinks) {
        int drinkNo = drinks[0];
        SQLiteOpenHelper starbuzzDatabaseHelper =
                                new StarbuzzDatabaseHelper(DrinkActivity.this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
            db.update("DRINK", drinkValues,
                                "_id = ?", new String[] {Integer.toString(drinkNo)});
            db.close();
            return true;
        } catch(SQLiteException e) {
            return false;
        }
    }

    ...
}
```

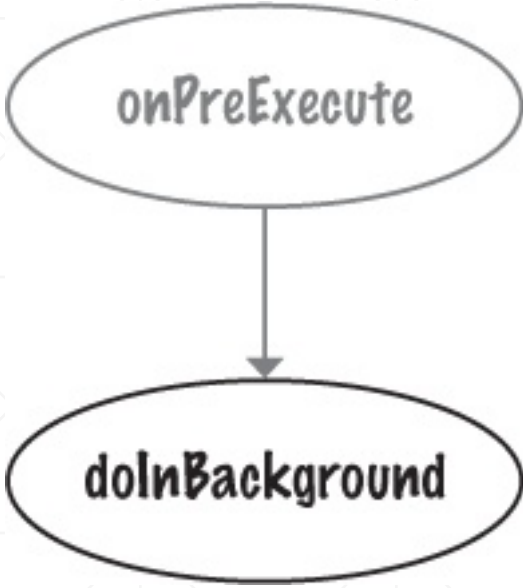
You change this to Integer to match the parameter of the doInBackground() method.

You change this to Boolean to match the return type of the doInBackground() method.

This code runs in a background thread.

This is an array of Integers, but we'll just include one item, the drink ID.

The update() method uses the drinkValues object that the onPreExecute() method created.



```
protected Boolean doInBackground(Integer... count) {
    for (int i = 0; i < count; i++) {
        publishProgress(i);
    }
}

protected void onProgressUpdate(Integer... progress) {
    setProgress(progress[0]);
}
```

This calls the onProgressUpdate() method, passing in a value of i.

We're not using the onProgressUpdate() method, so this is Void.

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {

    ...

}
```

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {

    ...

    protected void onPostExecute(Boolean success) {
        if (!success) {
            Toast toast = Toast.makeText(DrinkActivity.this,
                                         "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

This is Boolean, as our doInBackground() method returns a Boolean.

Pass the Toast the DrinkActivity context.

```
private class MyAsyncTask extends AsyncTask<Params, Progress, Result>

    protected void onPreExecute() {
        //Code to run before executing the task
    }

    protected Result doInBackground(Params... params) {
        //Code that you want to run in a background thread
    }

    protected void onProgressUpdate(Progress... values) {
        //Code that you want to run to publish the progress of your task
    }

    protected void onPostExecute(Result result) {
        //Code that you wan to run when the task is complete
    }
}
```

```
private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
    ...
    protected Boolean doInBackground(Integer... drinks) {
    ...
    }

    protected void onPostExecute(Boolean... success) {
    ...
    }
}
```

```
int drinkNo = (Integer)getIntent().getExtras().get(EXTRA_DRINKNO);
new UpdateDrinkTask().execute(drinkNo);
```

```
protected Boolean doInBackground(Integer... drinks) {
        ...
}
```

```
//Update the database when the checkbox is clicked
public void onFavoriteClicked(View view){
    int drinkNo = (Integer)getIntent().getExtras().get(EXTRA_DRINKNO);
    CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
    ContentValues drinkValues = new ContentValues();
    drinkValues.put("FAVORITE", favorite.isChecked());
    SQLiteOpenHelper starbuzzDatabaseHelper =
                            new StarbuzzDatabaseHelper(DrinkActivity.this);
    try {
        SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
        db.update("DRINK", drinkValues, "_id = ?", new String[] {Integer.toString(drinkNo)});
        db.close();
    } catch(SQLiteException e) {
        Toast toast = Toast.makeText(this, "Database unavailable", Toast.LENGTH_SHORT);
        toast.show();
    }
    new UpdateDrinkTask().execute(drinkNo);
}
```

All of this code is replaced by our AsyncTask.

Execute the AsyncTask and pass it the drink ID.

```
package com.hfad.starbuzz;

...
import android.os.AsyncTask;
public class DrinkActivity extends Activity {

...
    //Update the database when the checkbox is clicked
    public void onFavoriteClicked(View view){
        int drinkNo = (Integer)getIntent().getExtras().get(EXTRA_DRINKNO);
        new UpdateDrinkTask().execute(drinkNo);
    }

    //Inner class to update the drink.
    private class UpdateDrinkTask extends AsyncTask<Integer, Void, Boolean> {
        ContentValues drinkValues;

        protected void onPreExecute() {
            CheckBox favorite = (CheckBox)findViewById(R.id.favorite);
            drinkValues = new ContentValues();
            drinkValues.put("FAVORITE", favorite.isChecked());
        }
```

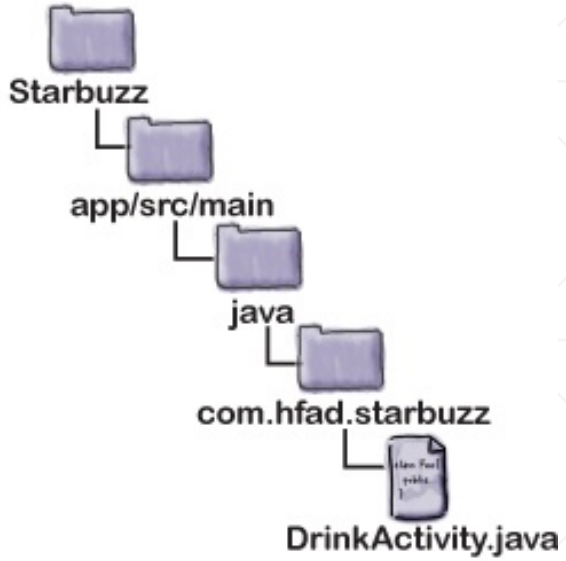
Import the AsyncTask class.

We don't need to change the onCreate() method, so we've left it out.

Execute the task.

Add the AsyncTask to the activity as an inner class.

Before the database code runs, put the value of the checkbox in the drinkValues ContentValues object.

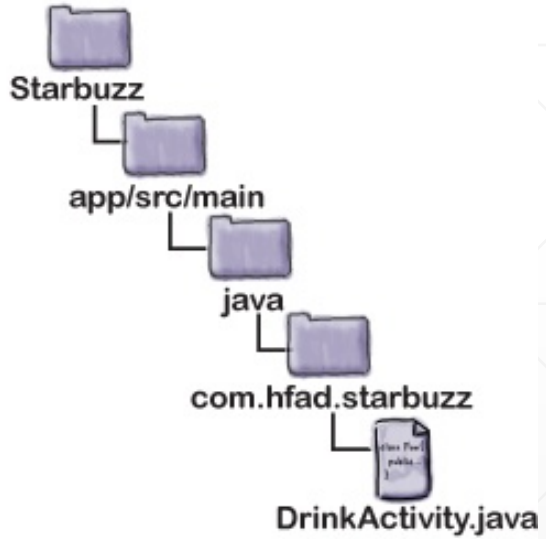


```
    protected Boolean doInBackground(Integer... drinks) {
        int drinkNo = drinks[0];
        SQLiteOpenHelper starbuzzDatabaseHelper =
                                new StarbuzzDatabaseHelper(DrinkActivity.this);
        try {
            SQLiteDatabase db = starbuzzDatabaseHelper.getWritableDatabase();
            db.update("DRINK", drinkValues,
                              "_id = ?", new String[] {Integer.toString(drinkNo)});
            db.close();
            return true;
        } catch(SQLiteException e) {
            return false;
        }
    }

    protected void onPostExecute(Boolean success) {
        if (!success) {
            Toast toast = Toast.makeText(DrinkActivity.this,
                              "Database unavailable", Toast.LENGTH_SHORT);
            toast.show();
        }
    }
}
```

Run the database code in a background thread.

Update the value of the FAVORITE column.

If the database code didn't run OK, display a message to the user.




onPreExecute
doInBackground
onProgressUpdate
onPostExecute

Chapter 13 – Services

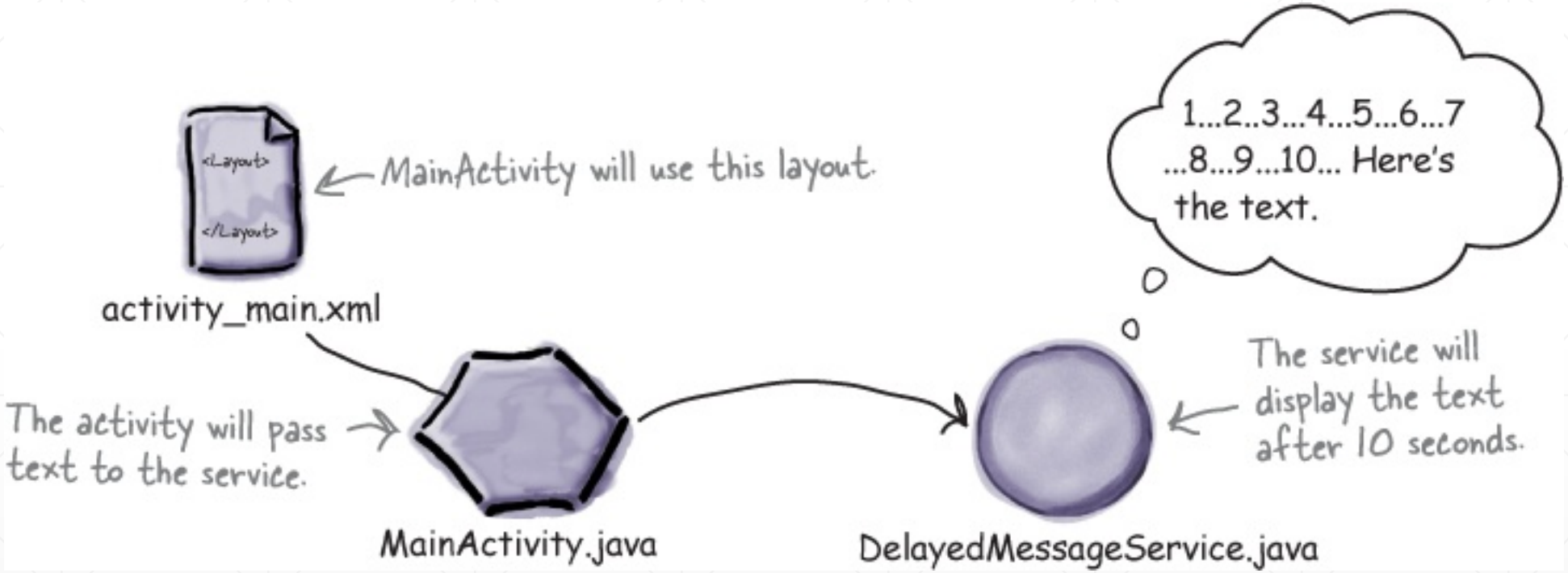
- Standard Services
- Bound Services

Bullet Points

- A service is a component that can perform tasks in the background. It doesn't have a user interface.
- A started service can run in the background indefinitely, even when the activity that started it is destroyed. Once the operation is done, it stops itself.
- You declare services in *AndroidManifest.xml* using the <service> element.
- You can create a simple started service by extending the IntentService class and overriding its onHandleIntent() method. The IntentServiceclass is designed for handling intents.
- You start a started service using the startService() method.
- If you override the IntentService onStartCommand() method, you must call its super implementation.
- You create a notification using a notification builder. You get your notification to start an activity using a pending intent. You then use Android's notification service to display the notification.
- A bound service is bound to another component such as an activity. The activity can interact with it and get results.

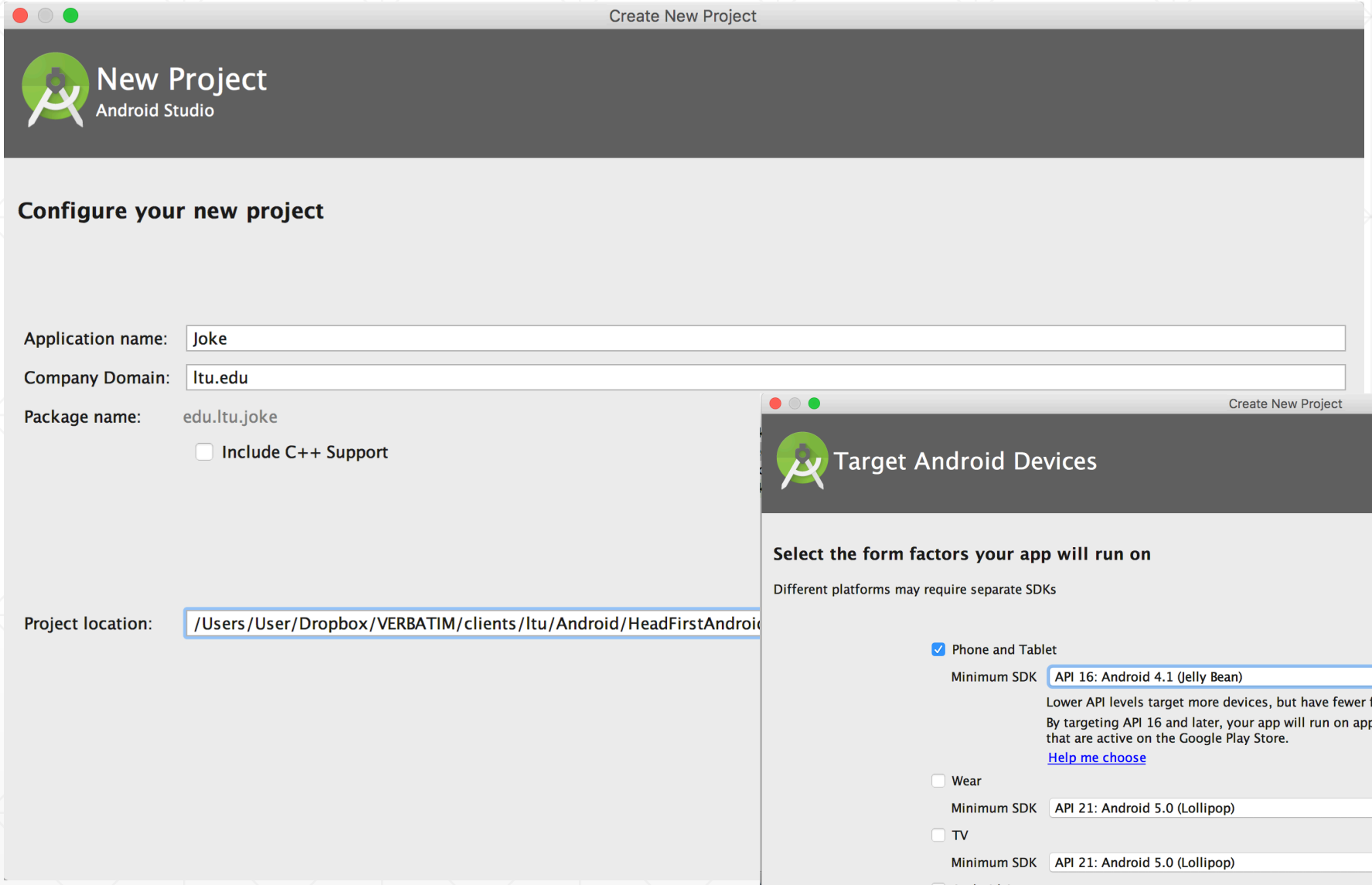
Bullet Points

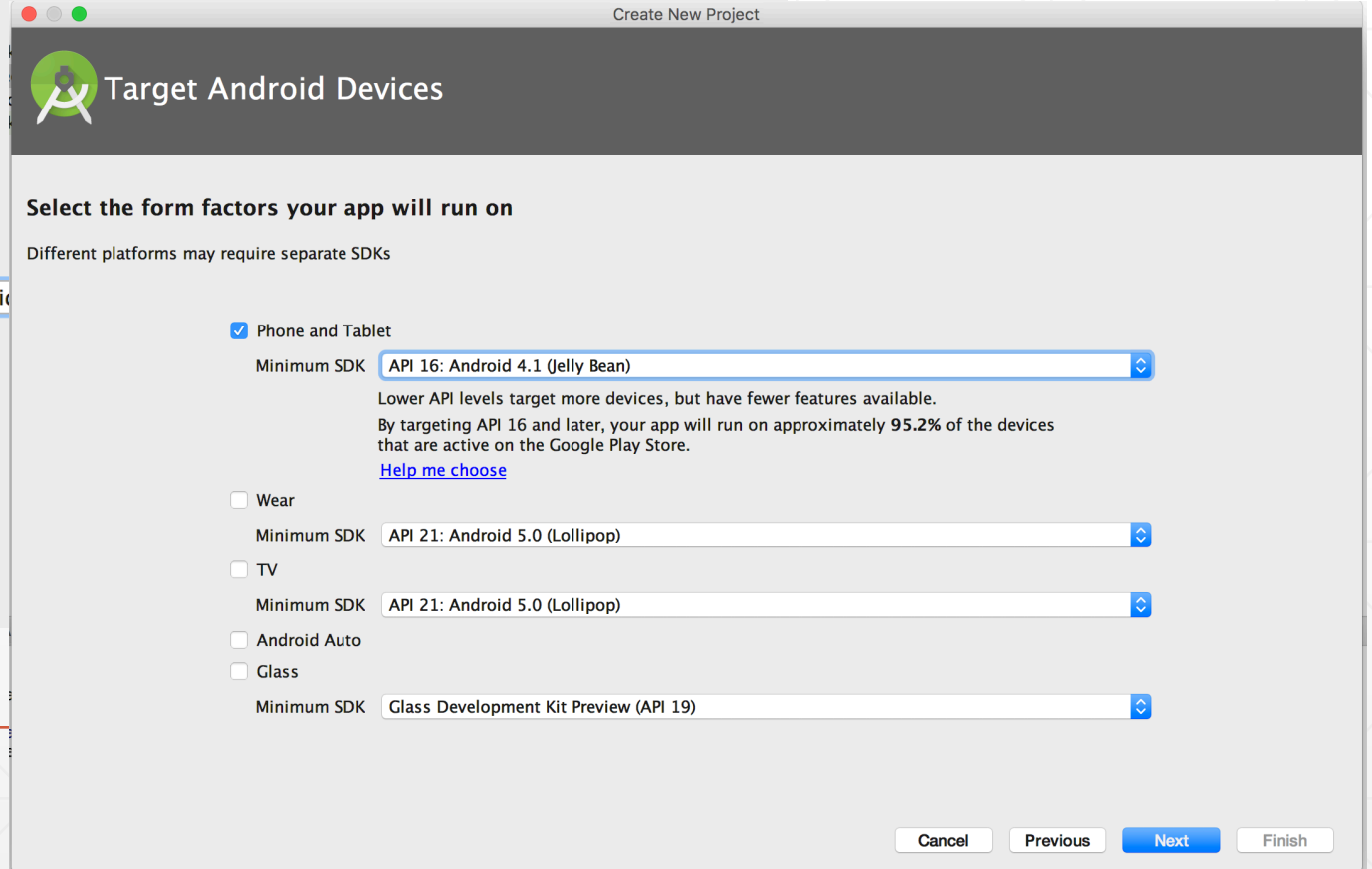
- You usually create a bound service by extending the Service class. You must define your own Binder object, and override the onBind() method. This is called when a component wants to bind to the service.
- The Service onCreate() method is called when the service is created. Use it for instantiation.
- The Service onDestroy() method is called when the service is about to be destroyed.
- You can use the Android location service to get the current location of the device. You create a LocationListener, and then register it with the location service. You can add criteria for how often the listener is notified of changes. When you use the device GPS, you need to add a permission for it in*AndroidManifest.xml*.
- To bind an activity to a service, you create a ServiceConnection. You override the onServiceConnected() method to get a reference to the service.
- You bind to the service using the bindService() method. You unbind from the service using the unbindService() method.

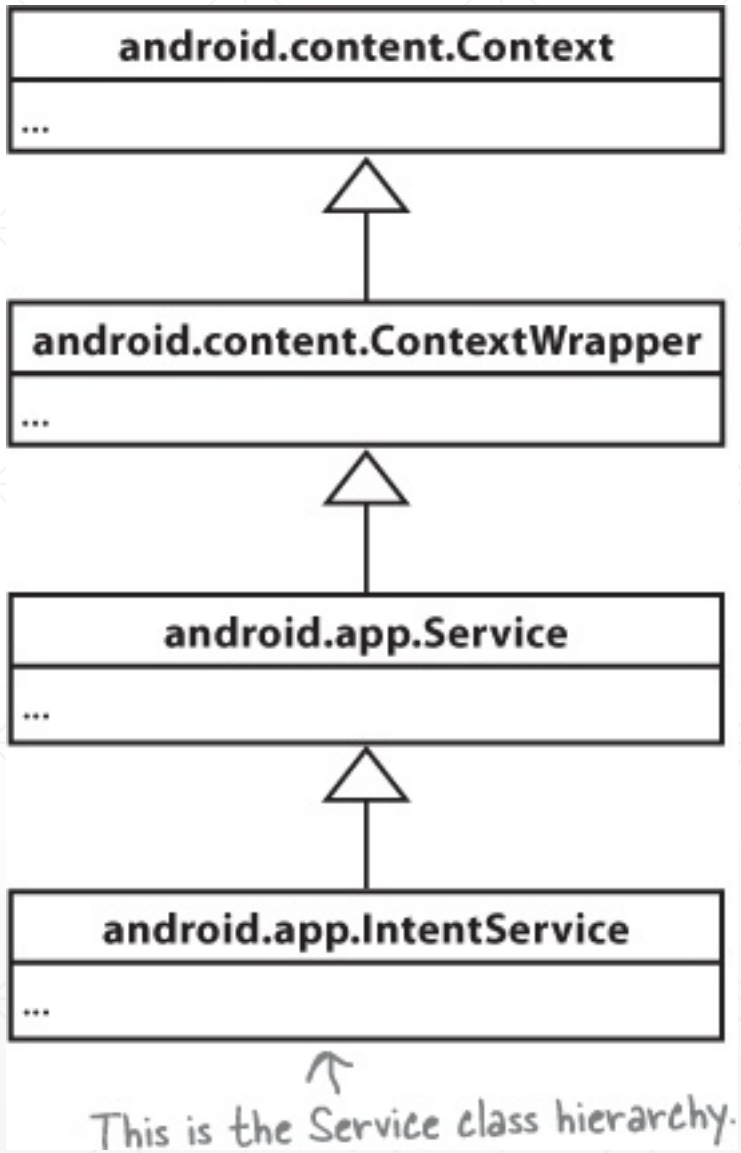

<Layout>
</Layout>
MainActivity will use this layout.
activity_main.xml
1...2...3...4...5...6...7
...8...9...10... Here's
the text.
The activity will pass
text to the service.
The service will
display the text
after 10 seconds.
MainActivity.java
DelayedMessageService.java

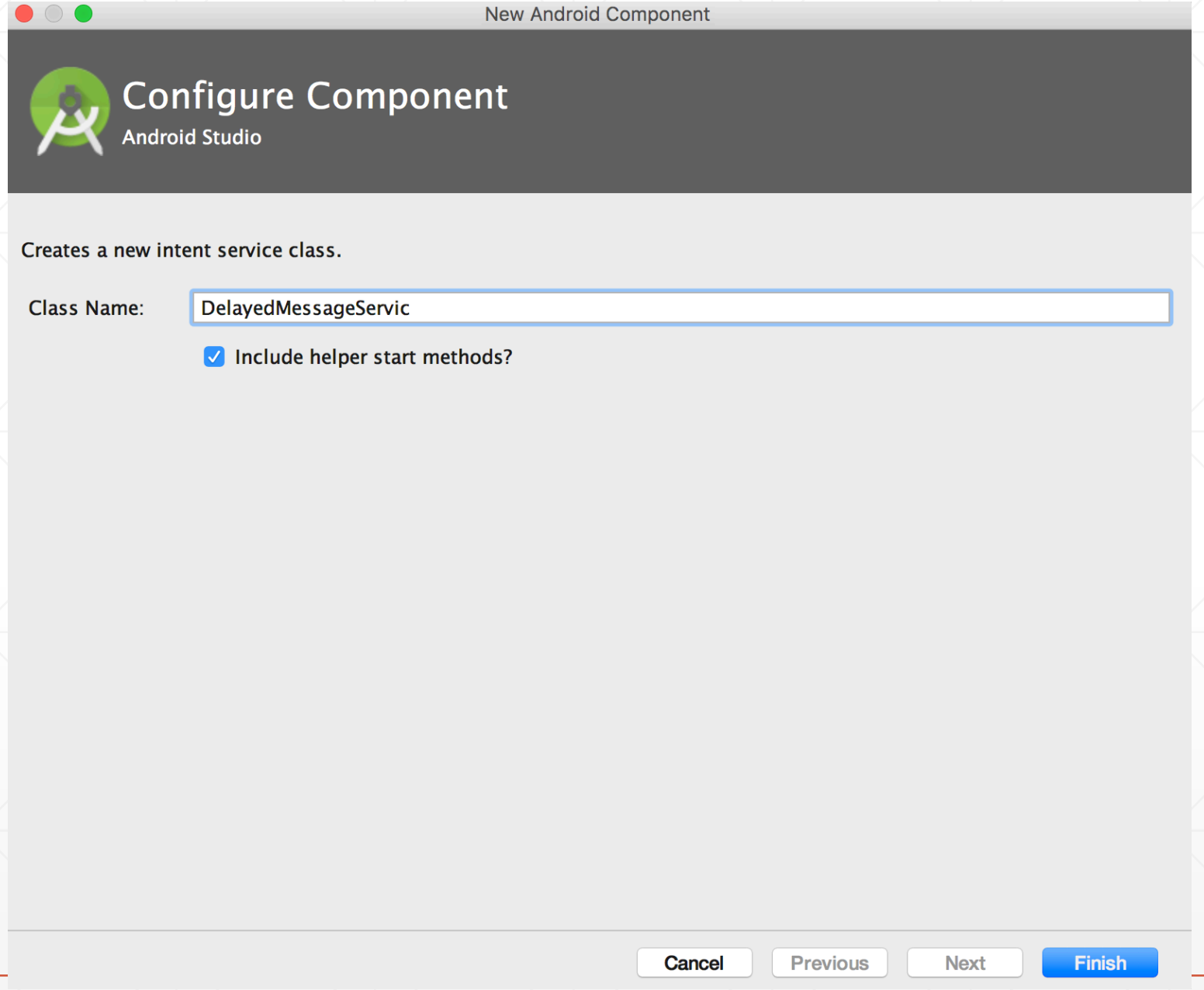
Standard Service

- Display the message in the log
- Display the message in a Toast
- Display message in a Notification


Create New Project
New Project
Android Studio
Configure your new project
Application name:
Joke
Company Domain:
ltu.edu
Package name:
edu.ltu.joke
Include C++ Support
Project location:
/Users/User/Dropbox/VERBATIM/clients/ltu/Android/HeadFirstAndroi


Create New Project
Target Android Devices
Select the form factors your app will run on
Different platforms may require separate SDKs
Phone and Tablet
Minimum SDK
API 16: Android 4.1 (Jelly Bean)
Lower API levels target more devices, but have fewer features available.
By targeting API 16 and later, your app will run on approximately 95.2% of the devices that are active on the Google Play Store.
Help me choose
Wear
Minimum SDK
API 21: Android 5.0 (Lollipop)
TV
Minimum SDK
API 21: Android 5.0 (Lollipop)
Android Auto
Glass
Minimum SDK
Glass Development Kit Preview (API 19)
Cancel
Previous
Next
Finish

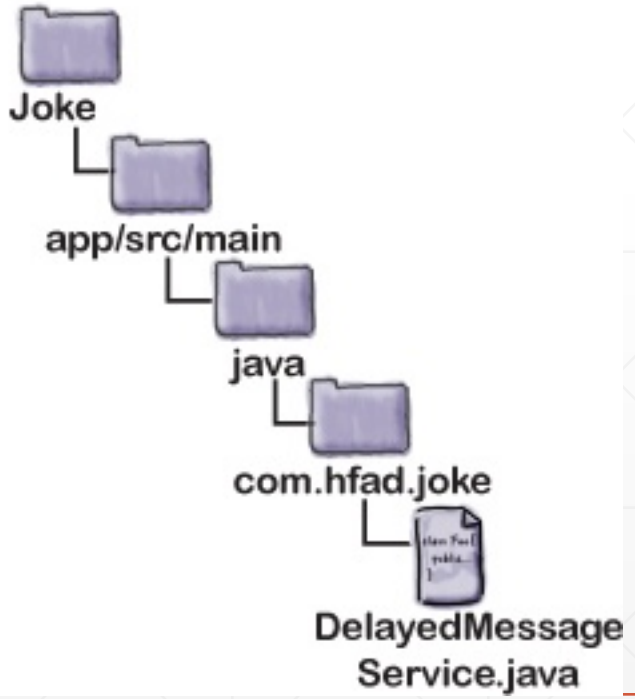

android.content.Context
...
android.content.ContextWrapper
...
android.app.Service
...
android.app.IntentService
...
This is the Service class hierarchy.

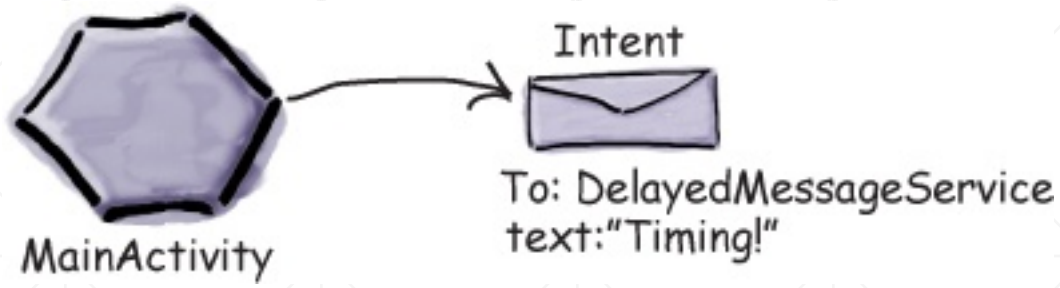

New Android Component
Configure Component
Android Studio
Creates a new intent service class.
Class Name:
DelayedMessageServic
Include helper start methods?
Cancel
Previous
Next
Finish

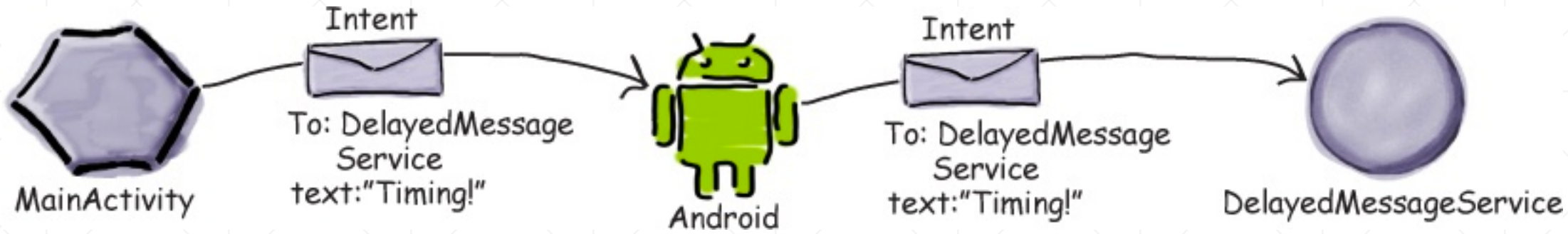
```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;

public class DelayedMessageService extends IntentService {

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

    @Override
    protected void onHandleIntent(Intent intent) {
        //Code to do something
    }
}
```

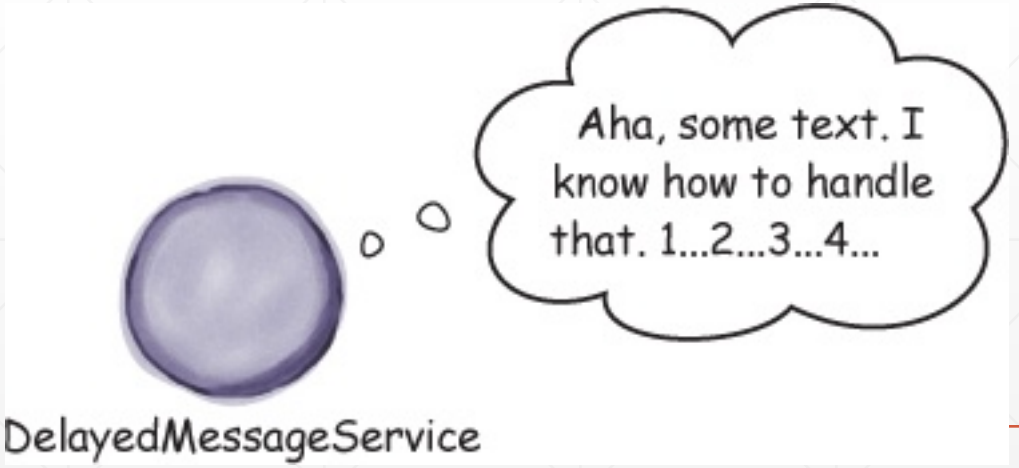
Extend the IntentService class.

Put the code you want the service to run in the onHandleIntent() method.



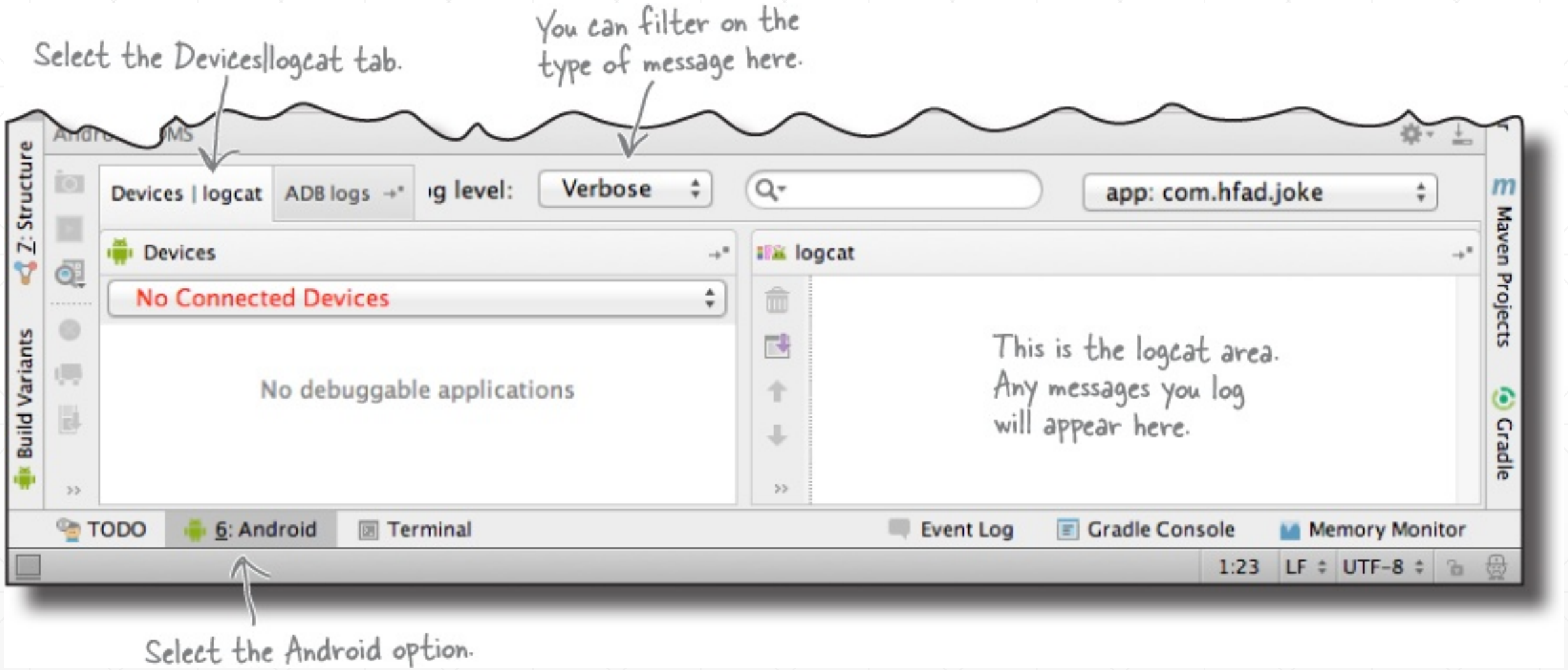

Intent
To: DelayedMessageService
text:"Timing!"
MainActivity


Intent
To: DelayedMessage
Service
text:"Timing!"
Intent
To: DelayedMessage
Service
text:"Timing!"
MainActivity
Android
DelayedMessageService


Aha, some text. I know how to handle that. 1...2...3...4...
DelayedMessageService

Log Type	Description
Log.v(String tag, String message)	Logs a verbose message.
Log.d(String tag, String message)	Logs a debug message.
Log.i(String tag, String message)	Logs an information message.
Log.w(String tag, String message)	Logs a warning message.
Log.e(String tag, String message)	Logs an error message.

Log.v("DelayedMessageService", "This is a message");


Select the Devices|logcat tab.
You can filter on the type of message here.
Devices | logcat
ADB logs
g level:
Verbose
app: com.hfad.joke
Devices
No Connected Devices
No debuggable applications
logcat
This is the logcat area.
Any messages you log
will appear here.
Z: Structure
Build Variants
Maven Projects
Gradle
TODO
6: Android
Terminal
Event Log
Gradle Console
Memory Monitor
1:23
LF
UTF-8
Select the Android option.

```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";

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

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }

    private void showText(final String text) {
        Log.v("DelayedMessageService", "The message is: " + text);
    }
}
```

Extend the IntentService class.

Use a constant to pass a message from the activity to the service.

Call the super constructor.

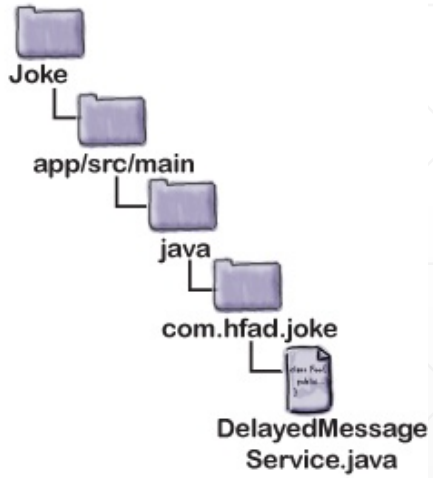
This method contains the code you want to run when the service receives an intent.

Wait 10 seconds.

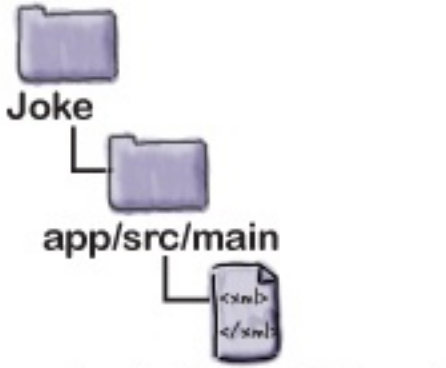
Get the text from the intent

Call the showText() method.

This logs a piece of text so we can see it in the logcat through Android Studio.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.joke" >
    <application
        ... >

        <activity
            ...
        </activity>

        <service
            android:name=".DelayedMessageService"
            android:exported="false" >
        </service>
    </application>
</manifest>
```



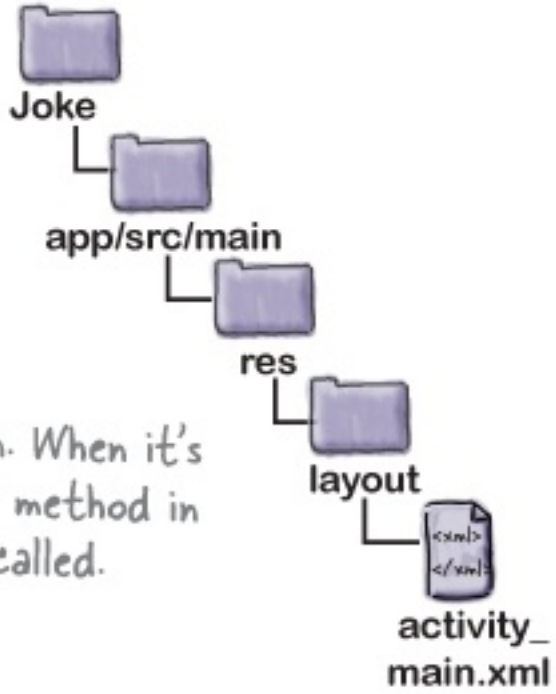
You declare a service in AndroidManifest.xml like this. Android Studio should do this for you automatically.

The service name has a . in front of it so that Android can combine it with the package name to derive the fully qualified class name.

```
<string name="button_response">Timing!</string>
<string name="button_text">What is the secret of comedy?</string>
```

We're using both these strings in the app.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_text"
        android:id="@+id/button"
        android:onClick="onClick"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```



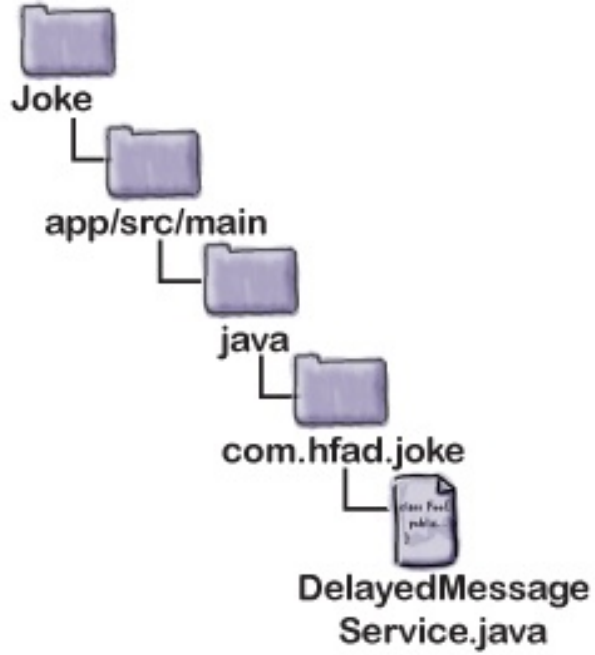
This creates a button. When it's clicked, the onClick() method in the activity will get called.

```
Intent intent = new Intent(this, DelayedMessageService.class);
startService(intent);
```

Starting a service is just like starting an activity, except you use startService() instead of startActivity().

```
package com.hfad.joke;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View view) {
        Intent intent = new Intent(this, DelayedMessageService.class);
        intent.putExtra(DelayedMessageService.EXTRA_MESSAGE,
                        getResources().getString(R.string.button_response));
        startService(intent);
    }
}
```

We're using these classes.



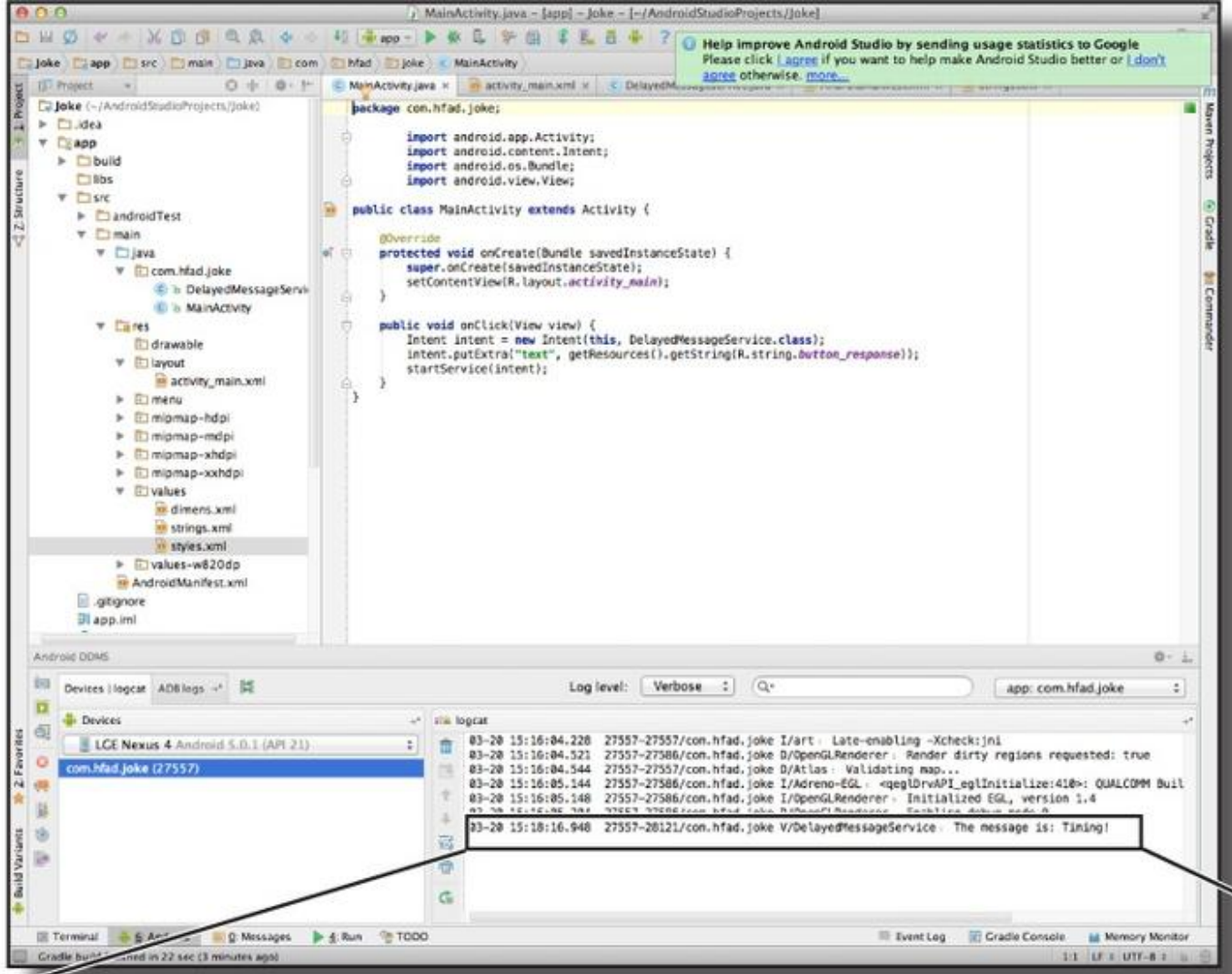
This will run when the button gets clicked.

Create the intent.

Add text to the intent.

Start the service.

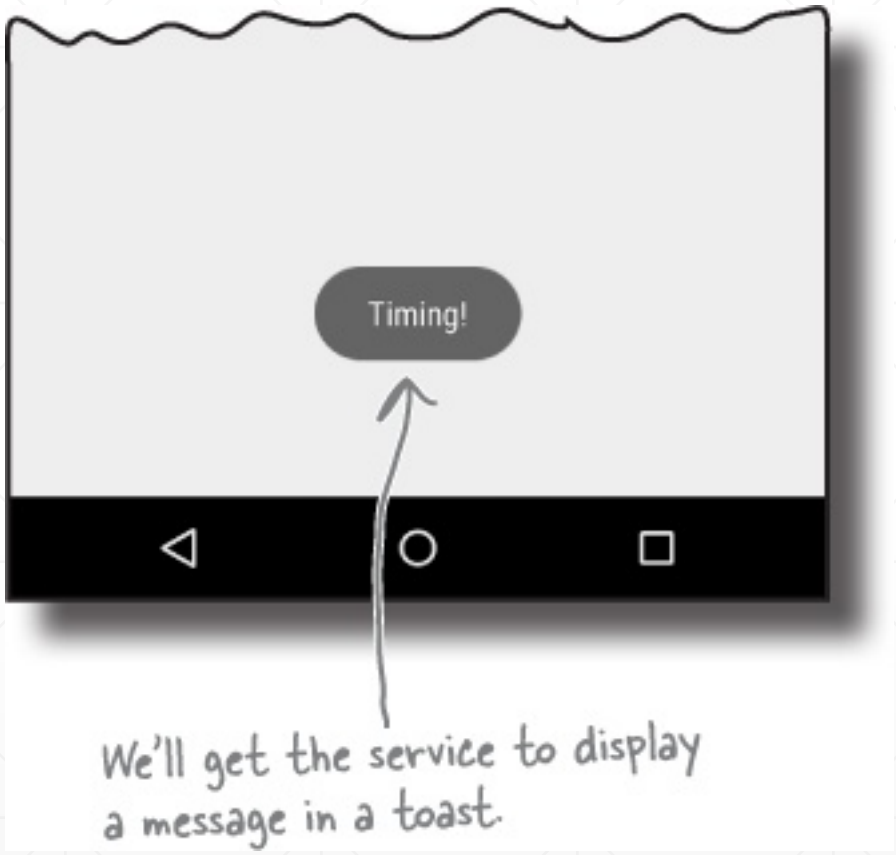

15:16
WHAT IS THE SECRET OF COMEDY?
Here's the button.



This is the logcat window.

```
03-20 15:18:16.948  27557-28121/com.hfad.joke V/DelayedMessageService: The message is: Timing!
```

After a 10-second delay, the message is displayed in the log.


Timing!
We'll get the service to display a message in a toast.

Screen updates require the main thread

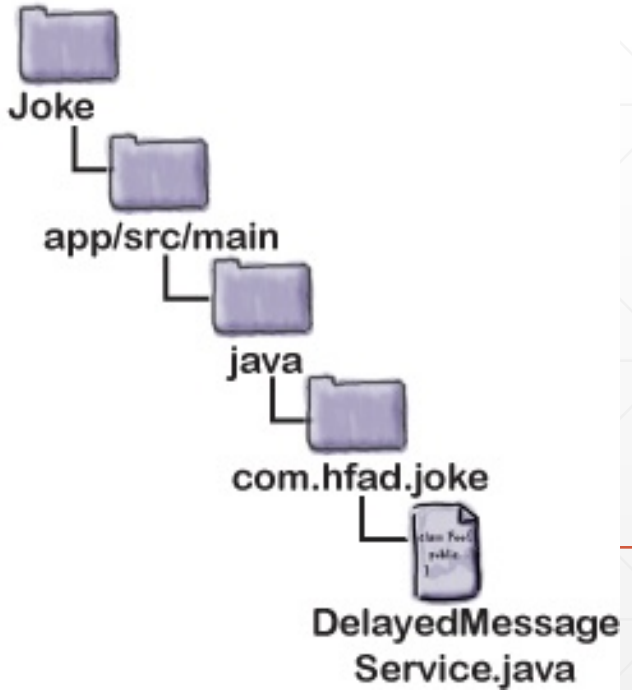
- Create a handler in the main method
- Use the Handler post() method in the service onHandleIntent() method to display

```
public class DelayedMessageService extends IntentService {

    private Handler handler;

    ...

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        handler = new Handler();
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        //Use the handler to post code to the main thread
    }
    ...
}
```

Add the handler as a private variable so different methods can access it.

This method runs on the main thread, so it creates a new handler on the main thread.

Call the IntentService onStartCommand() method.



```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;
import android.os.Handler;
import android.widget.Toast;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";
    private Handler handler;

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

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        handler = new Handler();
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }

    private void showText(final String text) {
        handler.post(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(getApplicationContext(), text, Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

We're using these extra classes.

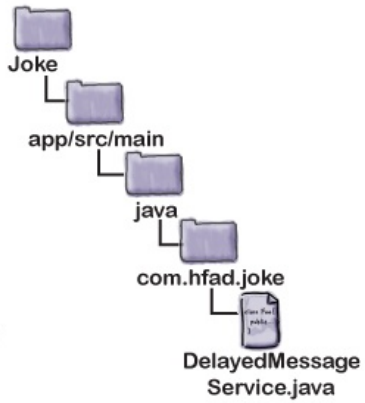
Add the handler as a new private variable.

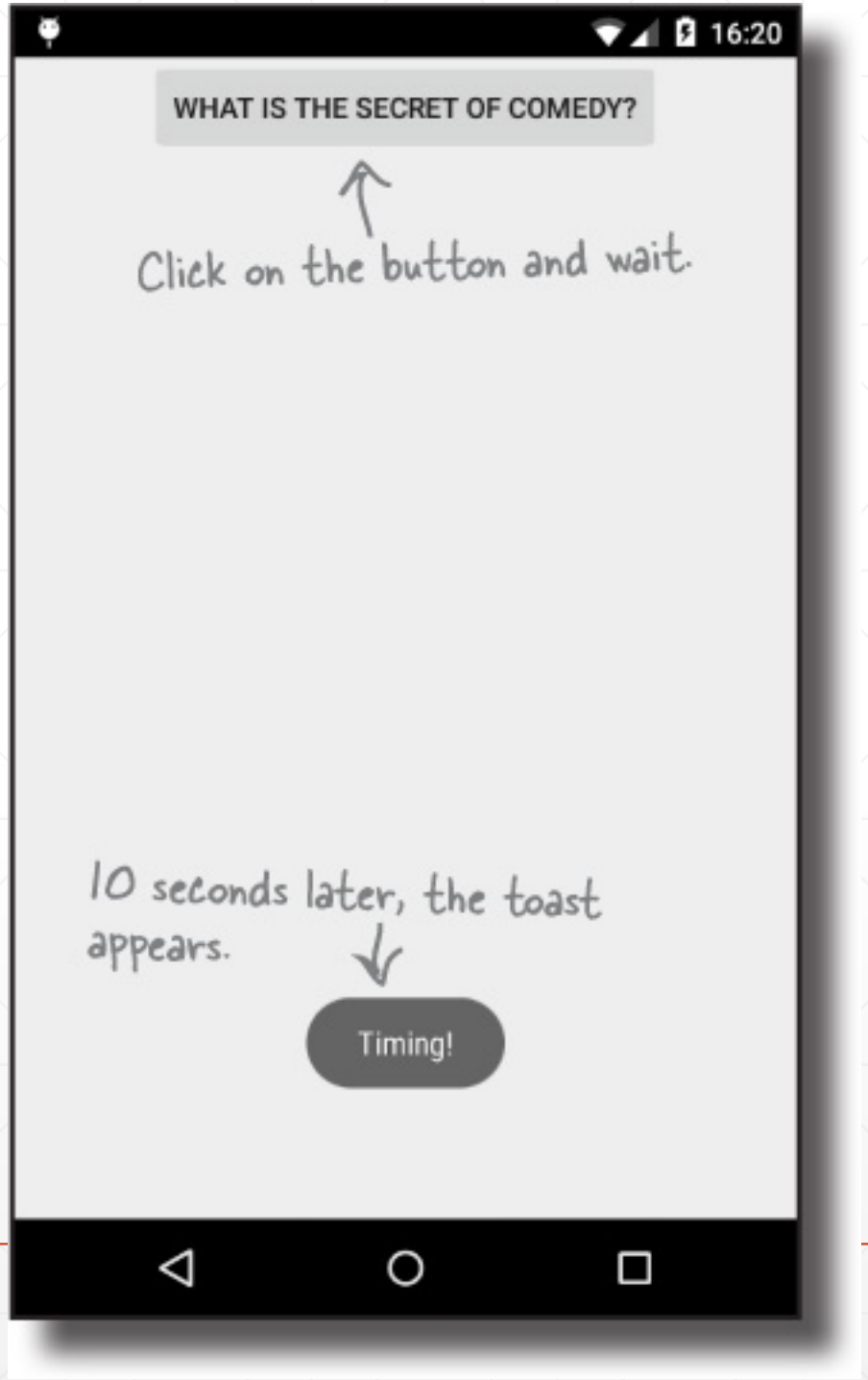
Create the handler on the main thread.

We're not changing this method.

Post the Toast code to the main thread using the handler.

This is the context you want to display the toast in. There's more about this on the next page.



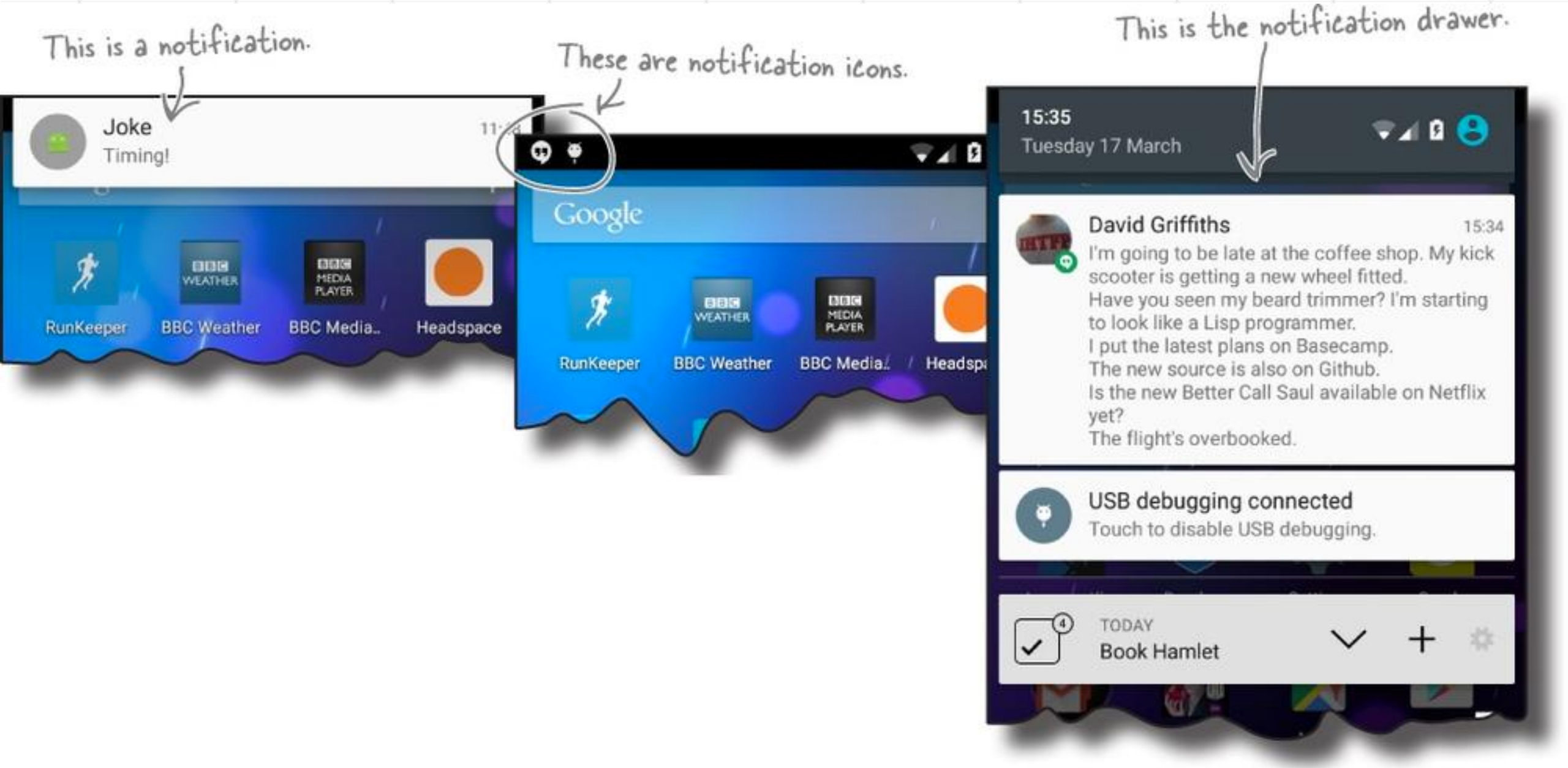

16:20
WHAT IS THE SECRET OF COMEDY?
Click on the button and wait.
10 seconds later, the toast appears.
Timing!

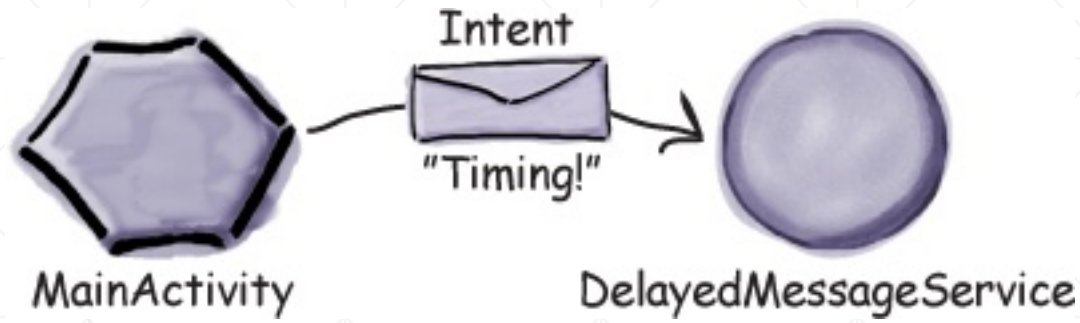
Bullet Points

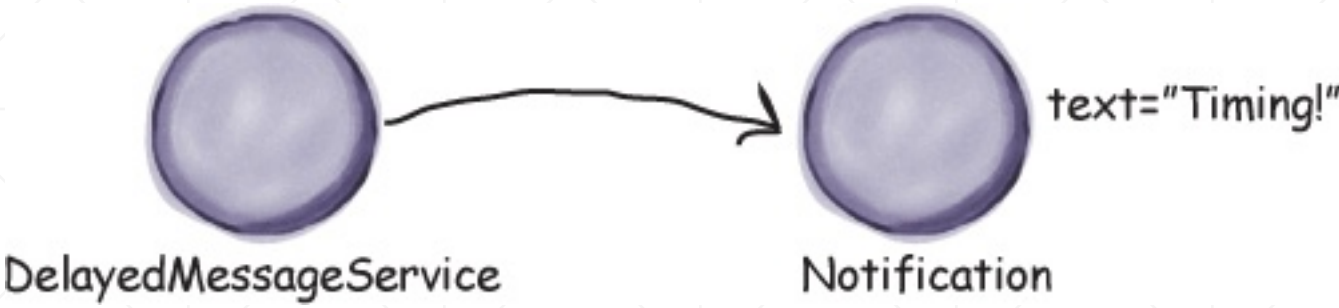
- A service is a component that can perform tasks in the background. It doesn't have a user interface.
- A started service can run in the background indefinitely, even when the activity that started it is destroyed. Once the operation is done, it stops itself.
- You declare services in *AndroidManifest.xml* using the <service> element.
- You can create a simple started service by extending the IntentService class and overriding its onHandleIntent() method. The IntentServiceclass is designed for handling intents.
- You start a started service using the startService() method.
- If you override the IntentService onStartCommand() method, you must call its super implementation.
- You create a notification using a notification builder. You get your notification to start an activity using a pending intent. You then use Android's notification service to display the notification.
- A bound service is bound to another component such as an activity. The activity can interact with it and get results.

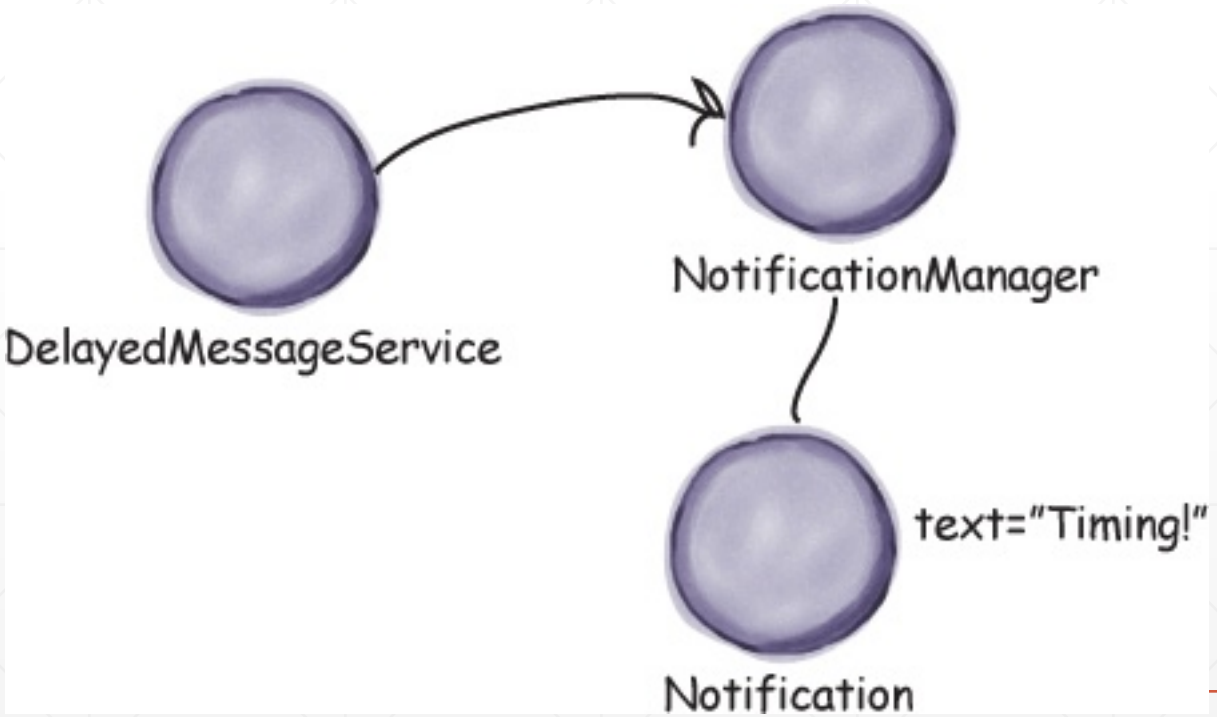
Bullet Points

- You usually create a bound service by extending the Service class. You must define your own Binder object, and override the onBind() method. This is called when a component wants to bind to the service.
- The Service onCreate() method is called when the service is created. Use it for instantiation.
- The Service onDestroy() method is called when the service is about to be destroyed.
- You can use the Android location service to get the current location of the device. You create a LocationListener, and then register it with the location service. You can add criteria for how often the listener is notified of changes. When you use the device GPS, you need to add a permission for it in*AndroidManifest.xml*.
- To bind an activity to a service, you create a ServiceConnection. You override the onServiceConnected() method to get a reference to the service.
- You bind to the service using the bindService() method. You unbind from the service using the unbindService() method.


This is a notification.
Joke
Timing!
RunKeeper
BBC Weather
BBC Media..
Headspace
These are notification icons.
Google
RunKeeper
BBC Weather
BBC Media..
This is the notification drawer.
15:35
Tuesday 17 March
David Griffiths
15:34
I'm going to be late at the coffee shop. My kick scooter is getting a new wheel fitted.
Have you seen my beard trimmer? I'm starting to look like a Lisp programmer.
I put the latest plans on Basecamp.
The new source is also on Github.
Is the new Better Call Saul available on Netflix yet?
The flight's overbooked.
USB debugging connected
Touch to disable USB debugging.
TODAY
Book Hamlet


Intent
"Timing!"
MainActivity
DelayedMessageService


DelayedMessageService
Notification
text="Timing!"


NotificationManager
DelayedMessageService
text="Timing!"
Notification

This displays a small notification icon—in this case, the mipmap called ic_launcher.

```
Notification notification =  new Notification.Builder(this)
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle(getString(R.string.app_name))
        .setContentText(text)
        .setAutoCancel(true)
        .setPriority(Notification.PRIORITY_MAX)
        .setDefaults(Notification.DEFAULT_VIBRATE)
        .build();
```

Set the title and text.

Make the notification disappear when clicked.

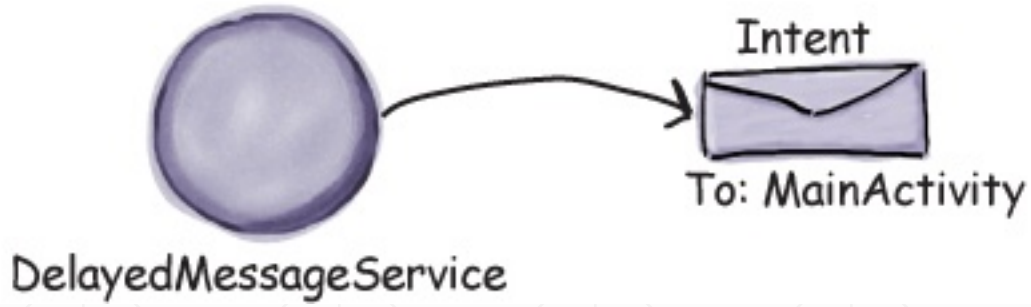
Give it a maximum priority and set it to vibrate to get a large "heads up" notification.

Getting notification to start an activity

- Create an Explicit Intent
- Pass the Intent to the TaskStackBuilder
- Get the Pending Intent from the TaskStackBuilder
- Add the Intent to the Notification

This is a normal intent that starts MainActivity.

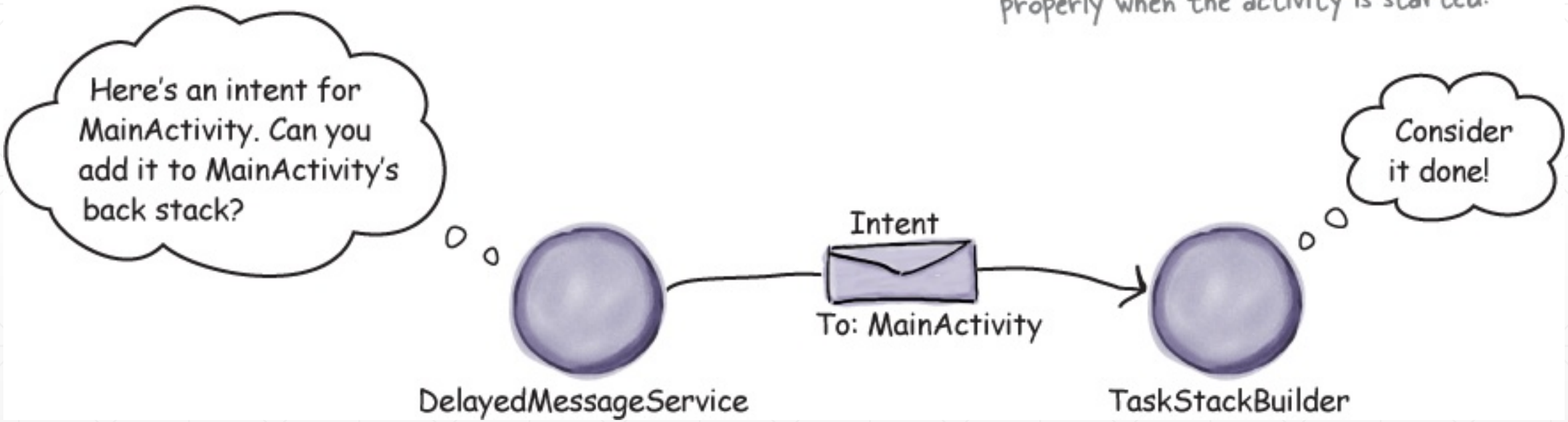
```
Intent intent = new Intent(this, MainActivity.class);
```



Create a TaskStackBuilder.

```
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(MainActivity.class);
stackBuilder.addNextIntent(intent);
```

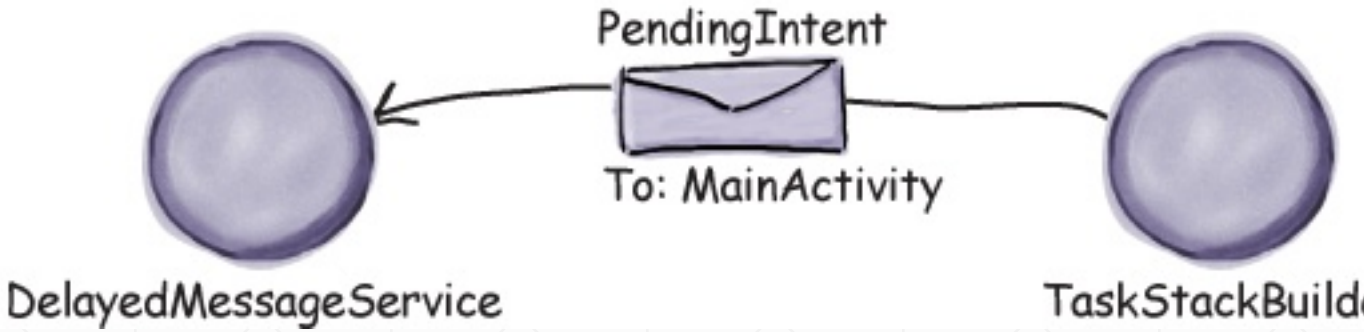
These lines make the back button work properly when the activity is started.



FLAG	Description
FLAG_CANCEL_CURRENT	If a matching pending intent already exists, cancel it before generating a new one.
FLAG_NO_CREATE	If a matching pending intent doesn’t already exist, don’t create one and return null.
FLAG_ONE_SHOT	The pending intent can only be used once.
FLAG_UPDATE_CURRENT	If a matching pending intent already exists, keep it and replace its extra data with the contents of the new intent.

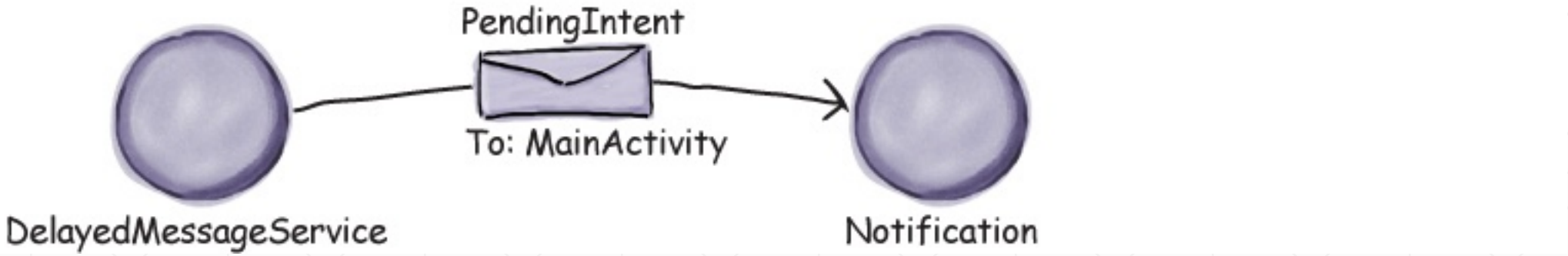
```
PendingIntent pendingIntent =
        stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
```

This creates the pending intent.



```
notification.setContentIntent(pendingIntent);
```

Add the pending intent to the notification so that MainActivity starts when it's clicked.



This is an ID we'll use for the notification.

```
public static final int NOTIFICATION_ID = 5453;
...
NotificationManager notificationManager =
            (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFICATION_ID, notification);
```

This is how you access Android's notification service.

Use the notification service to display the notification we created.

```
package com.hfad.joke;

import android.app.IntentService;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.TaskStackBuilder;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.widget.Toast;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";
    private Handler handler;
    public static final int NOTIFICATION_ID = 5453;

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

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }
```

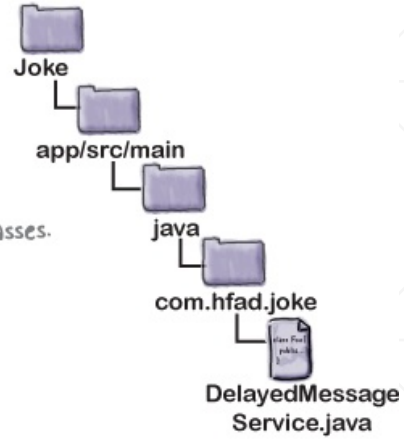
We're using these extra classes.

We're no longer displaying a Toast, so we don't need these imports.

We no longer need a Handler.

This is used to identify the notification. It could be any number, we just decided on 5453.

We're not changing this method.



```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    handler = new Handler();
    return super.onStartCommand(intent, flags, startId);
}
```

We're no longer using a Handler, so we don't need this method.

```
private void showText(final String text) {
    handler.post(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(getApplicationContext(), text, Toast.LENGTH_LONG).show();
        }
    });
```

We're no longer displaying the message using a Toast.

```
    Intent intent = new Intent(this, MainActivity.class);
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    stackBuilder.addParentStack(MainActivity.class);
    stackBuilder.addNextIntent(intent);
    PendingIntent pendingIntent =
            stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT
            );
    Notification notification = new Notification.Builder(this)
            .setSmallIcon(R.mipmap.ic_launcher)
            .setContentTitle(getString(R.string.app_name))
            .setAutoCancel(true)
            .setPriority(Notification.PRIORITY_MAX)
            .setDefaults(Notification.DEFAULT_VIBRATE)
            .setContentIntent(pendingIntent)
            .setContentText(text)
            .build();
    NotificationManager notificationManager =
            (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.notify(NOTIFICATION_ID, notification);
}
```

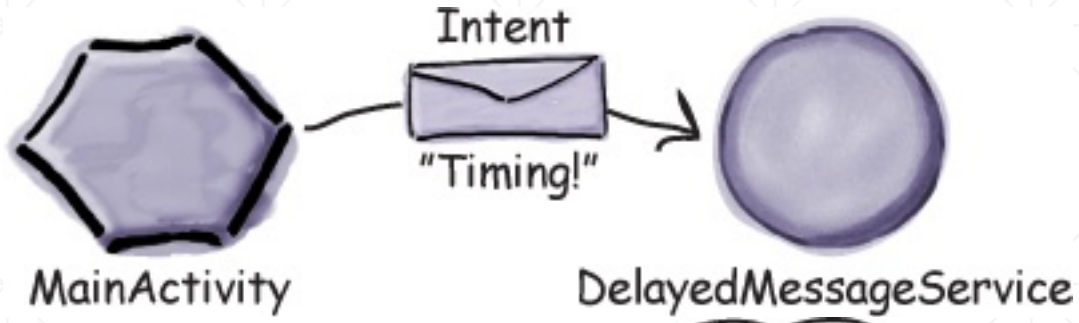
Create an intent.

Use a TaskStackBuilder to make the back button play nicely and create the pending intent.

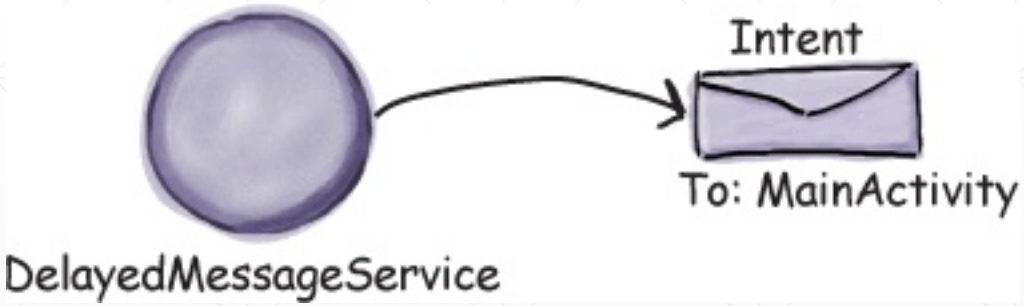
Build the notification.

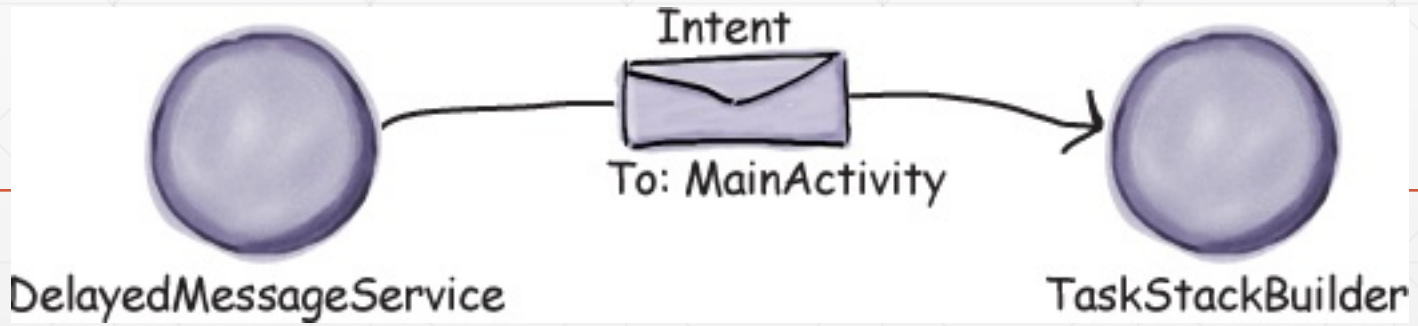
Display the notification using the Android notification service.

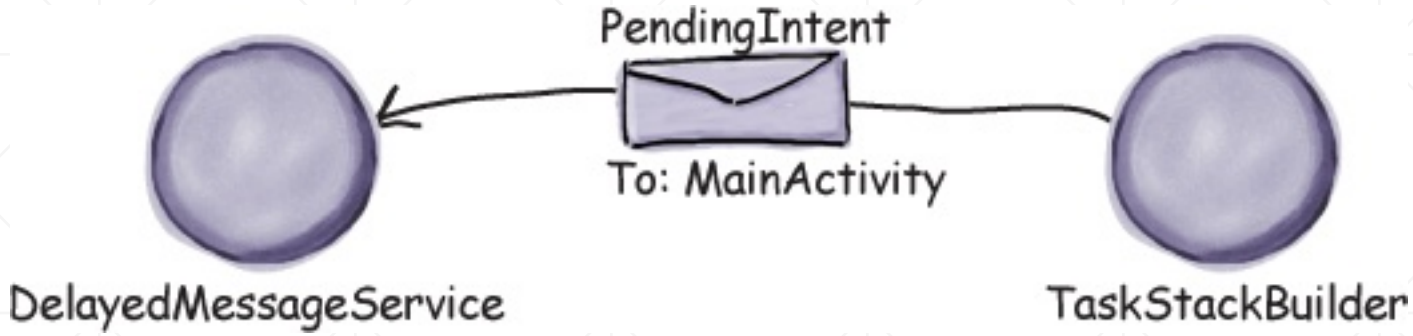
}

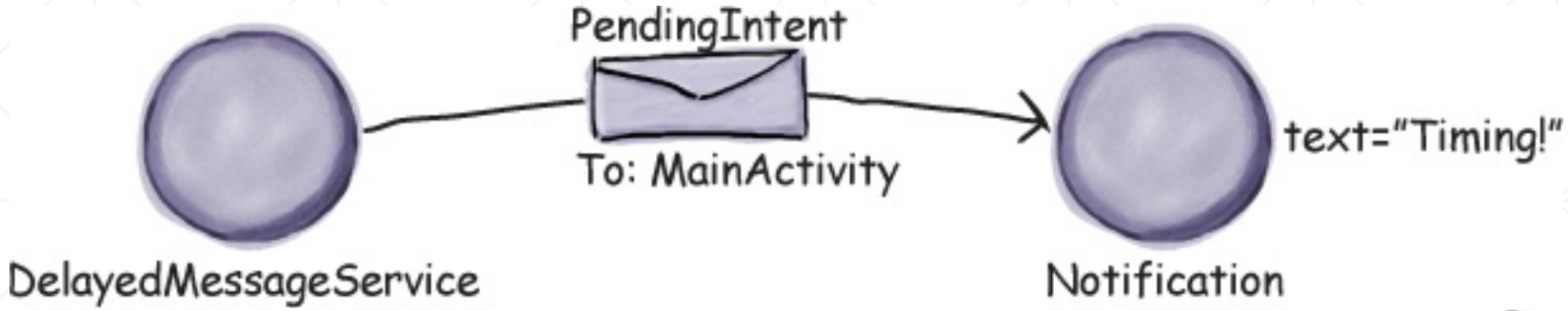

Intent
"Timing!"
MainActivity
DelayedMessageService

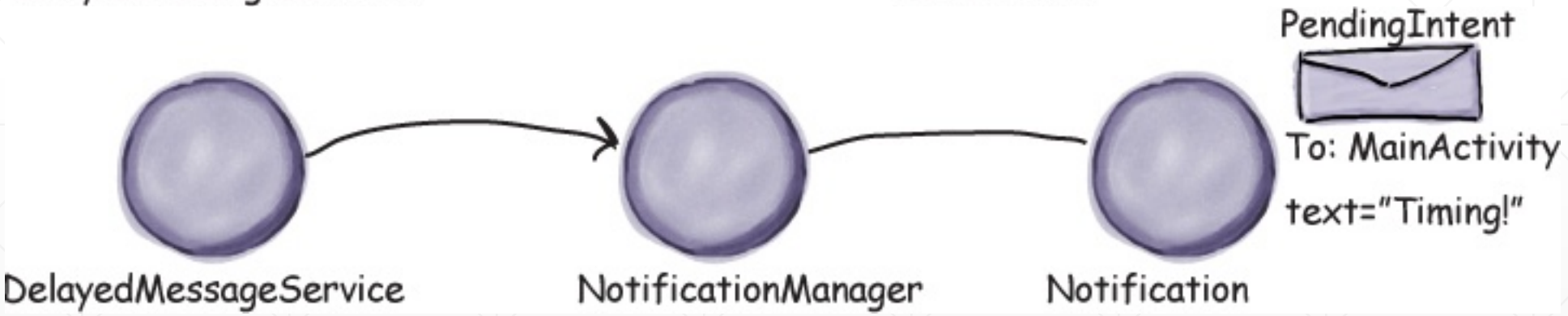

1...2...3...4...
DelayedMessageService

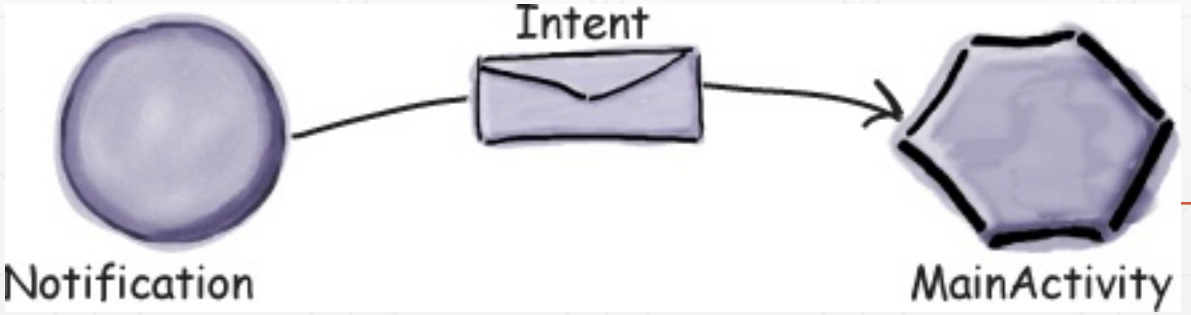

Intent
To: MainActivity
DelayedMessageService

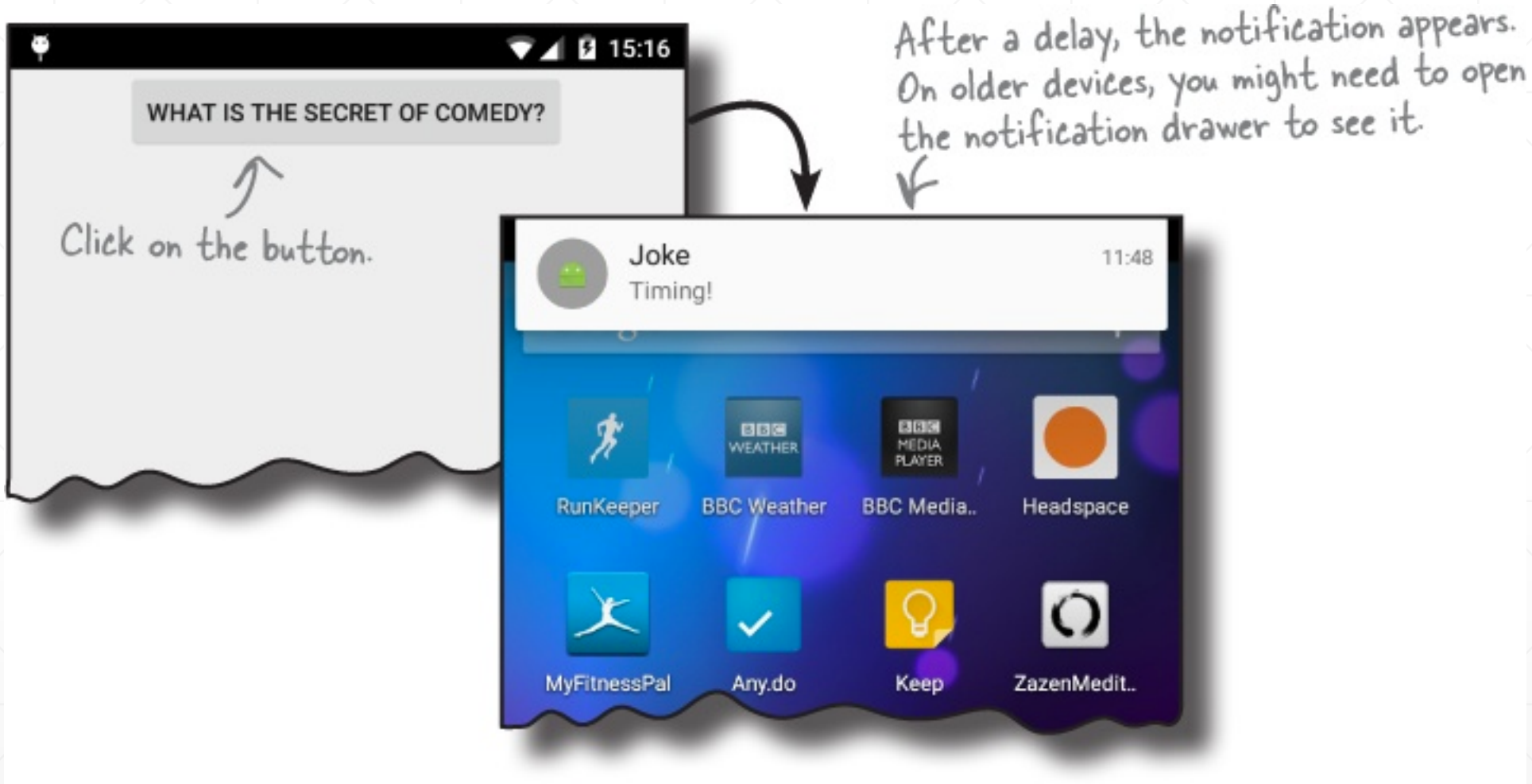

Intent
To: MainActivity
DelayedMessageService
TaskStackBuilder


PendingIntent
To: MainActivity
DelayedMessageService
TaskStackBuilder


PendingIntent
To: MainActivity
text="Timing!"
DelayedMessageService
Notification

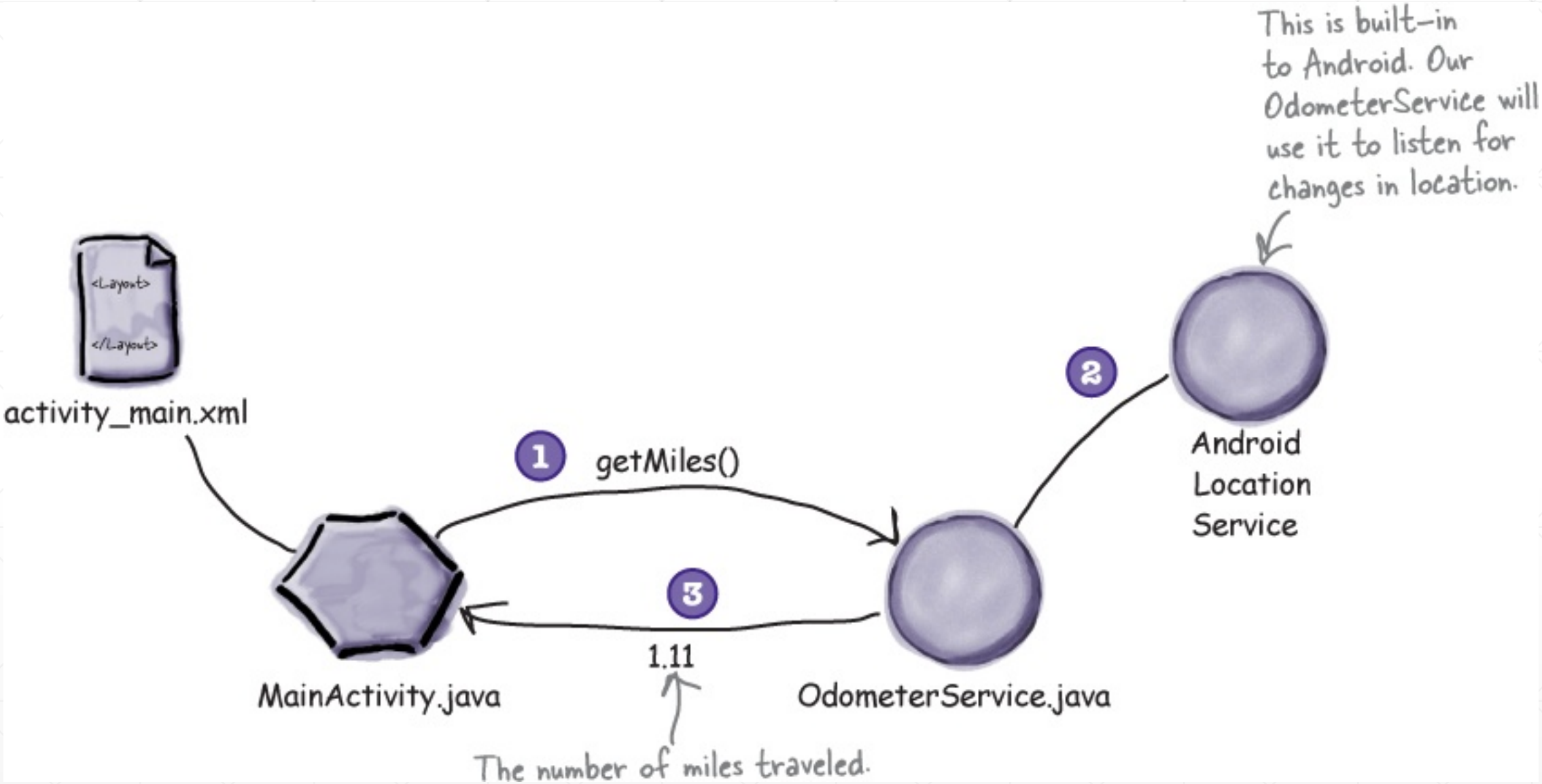

PendingIntent
To: MainActivity
text="Timing!"
DelayedMessageService
NotificationManager
Notification


Intent
Notification
MainActivity


15:16
WHAT IS THE SECRET OF COMEDY?
Click on the button.
After a delay, the notification appears.
On older devices, you might need to open
the notification drawer to see it.
Joke
11:48
Timing!
RunKeeper
BBC Weather
BBC Media..
Headspace
MyFitnessPal
Any.do
Keep
ZazenMedit..

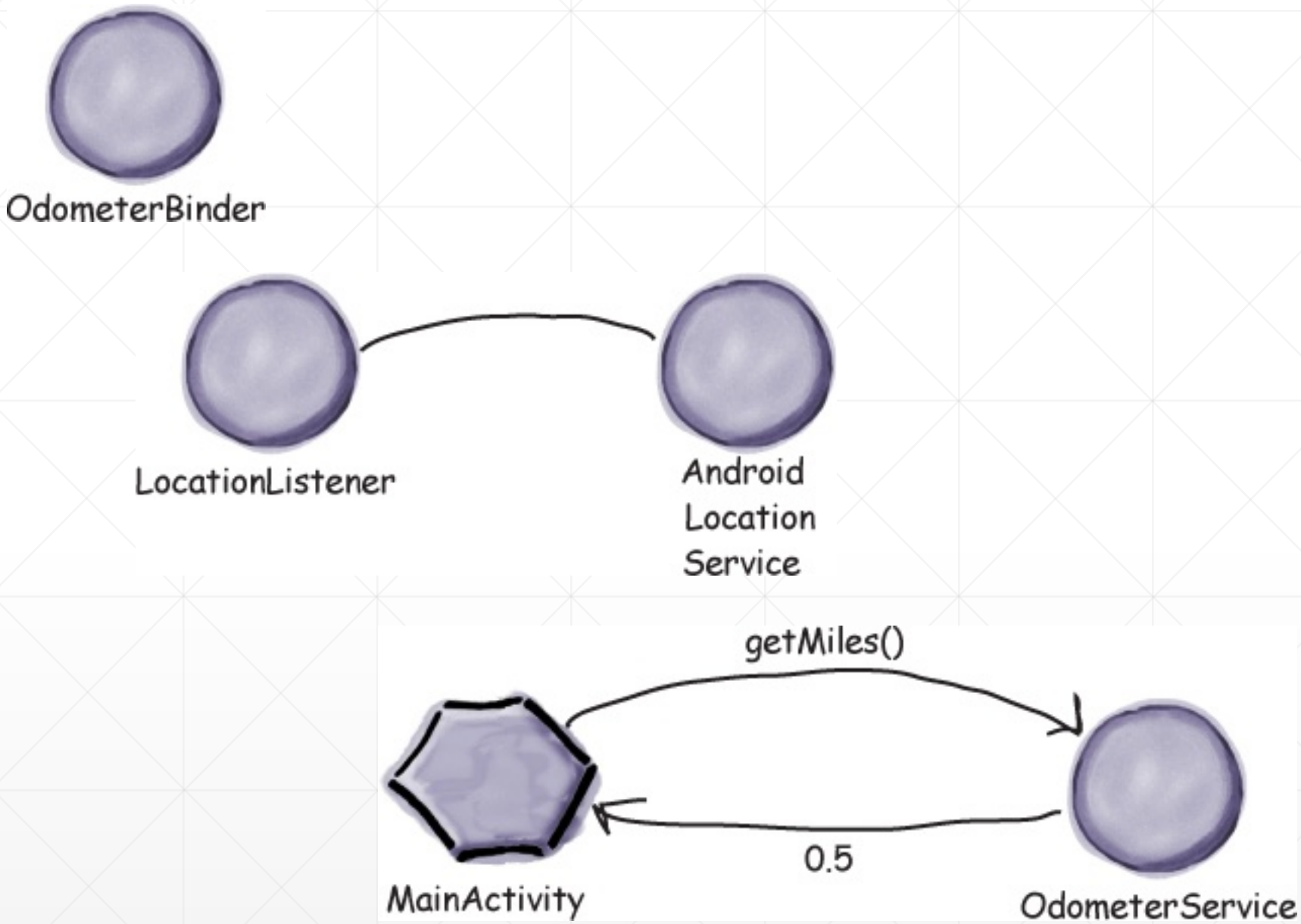
Clicking on the notification starts MainActivity, just as we wanted.

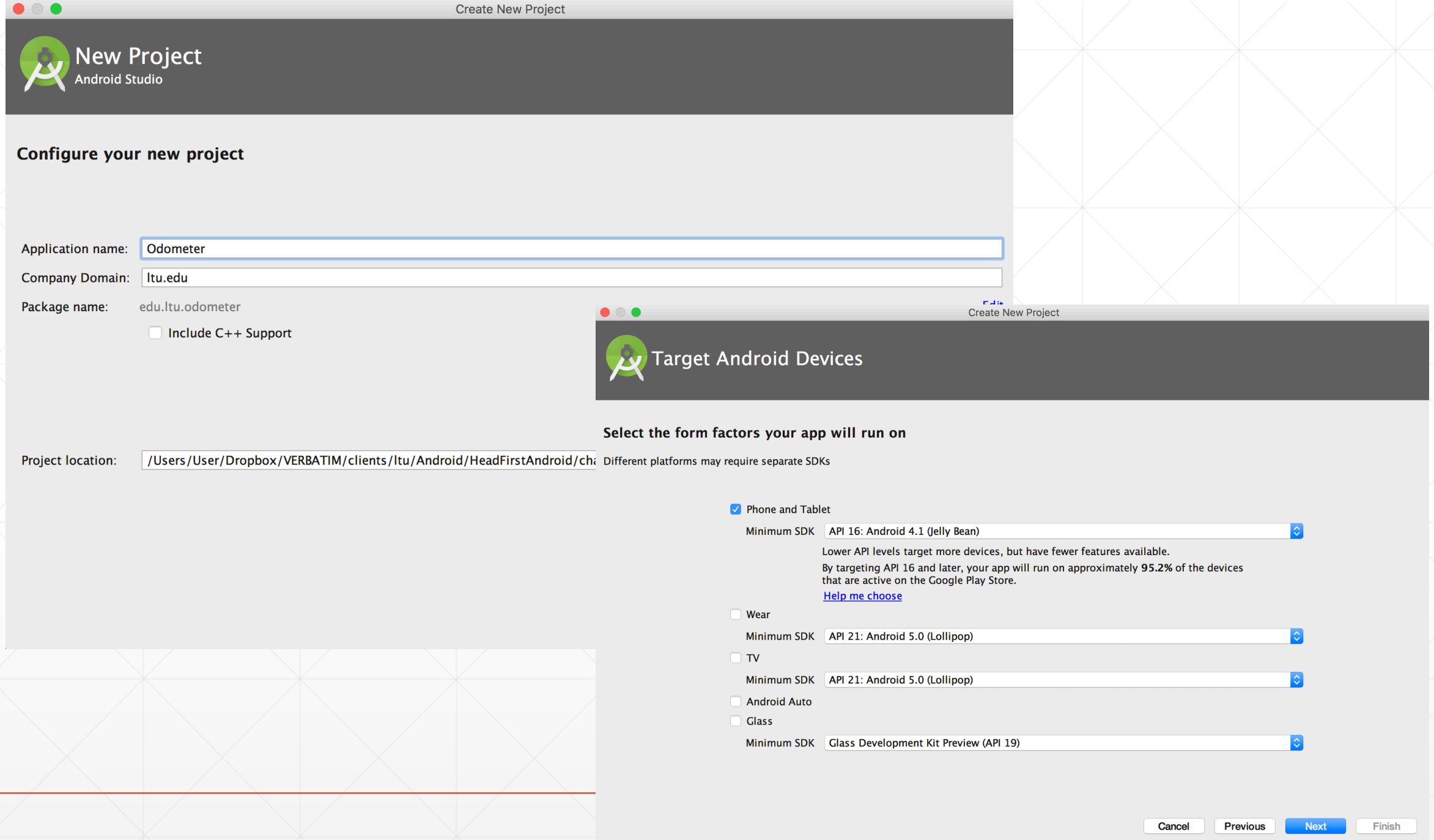

Android Emulator - Nexus_5X_API_23_hax...
3:39
0.00 miles

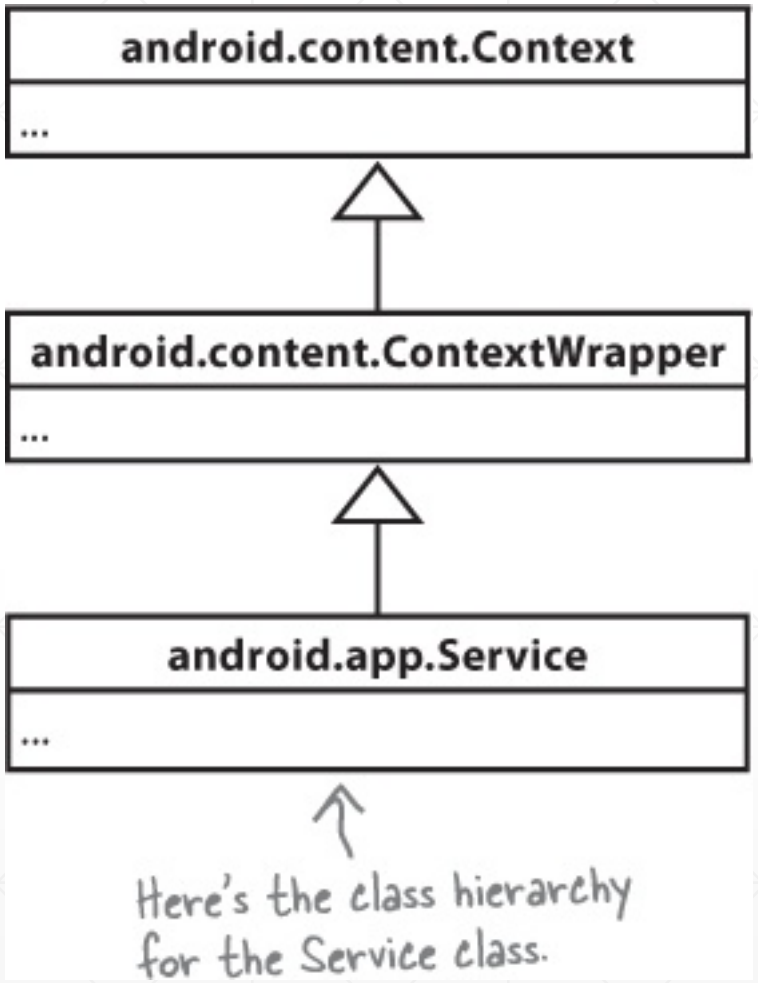

This is built-in to Android. Our OdometerService will use it to listen for changes in location.
<Layout>
</Layout>
activity_main.xml
1
getMiles()
2
Android Location Service
3
1.11
MainActivity.java
OdometerService.java
The number of miles traveled.

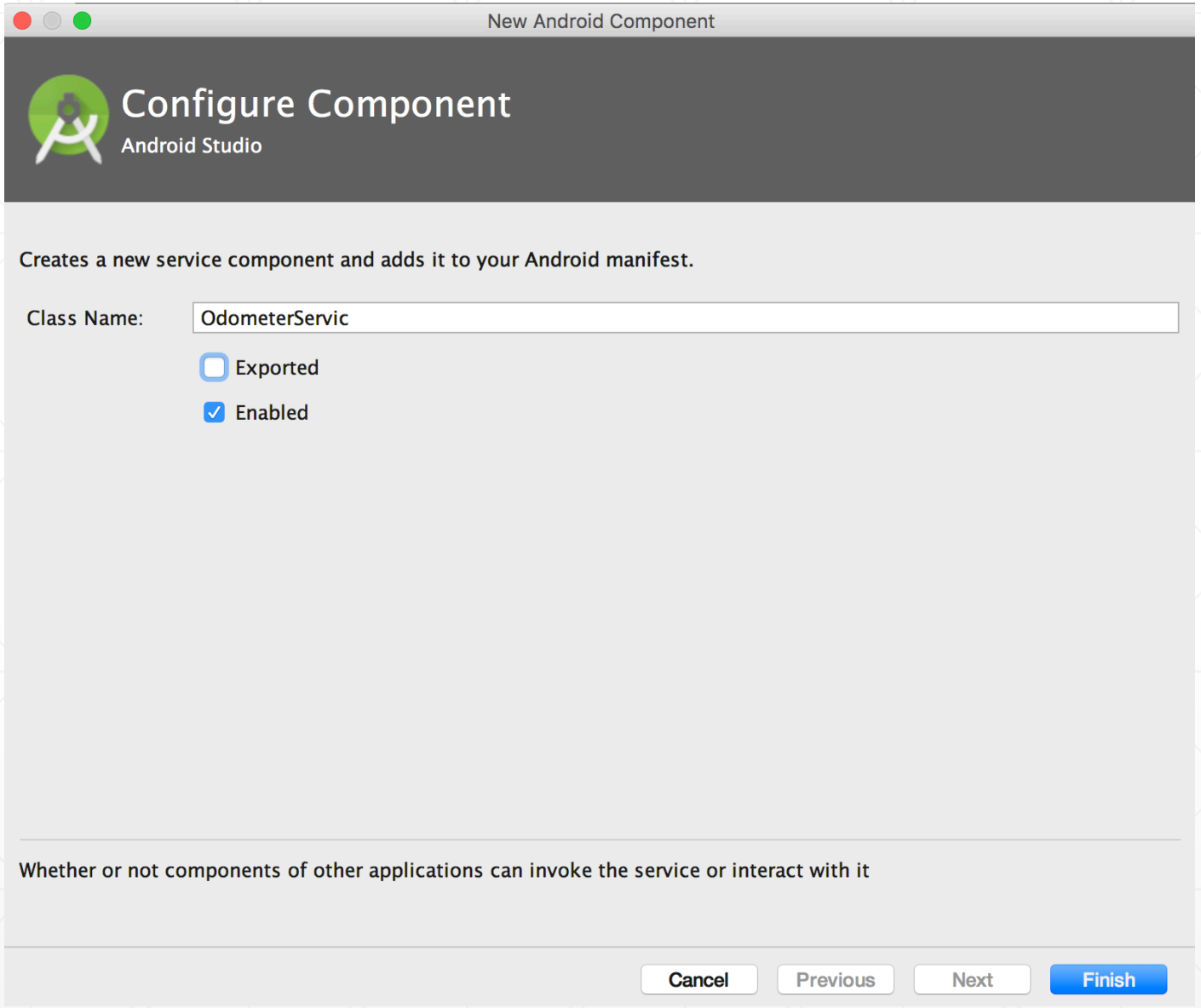
Steps to create OdometerService

- Binder
- Location
- getMiles()


OdometerBinder
LocationListener
Android
Location
Service
getMiles()
0.5
MainActivity
OdometerService


Create New Project
New Project
Android Studio
Configure your new project
Application name:
Odometer
Company Domain:
ltu.edu
Package name:
edu.ltu.odometer
Include C++ Support
Project location:
/Users/User/Dropbox/VERBATIM/clients/ltu/Android/HeadFirstAndroid/cha
Create New Project
Target Android Devices
Select the form factors your app will run on
Different platforms may require separate SDKs
Phone and Tablet
Minimum SDK
API 16: Android 4.1 (Jelly Bean)
Lower API levels target more devices, but have fewer features available.
By targeting API 16 and later, your app will run on approximately 95.2% of the devices that are active on the Google Play Store.
Help me choose
Wear
Minimum SDK
API 21: Android 5.0 (Lollipop)
TV
Minimum SDK
API 21: Android 5.0 (Lollipop)
Android Auto
Glass
Minimum SDK
Glass Development Kit Preview (API 19)
Cancel
Previous
Next
Finish

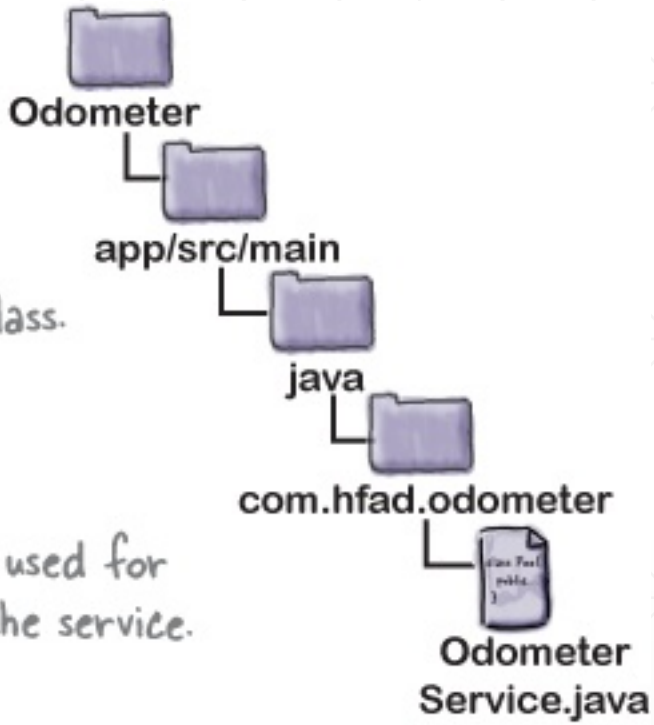

android.content.Context
...
android.content.ContextWrapper
...
android.app.Service
...
Here's the class hierarchy
for the Service class.

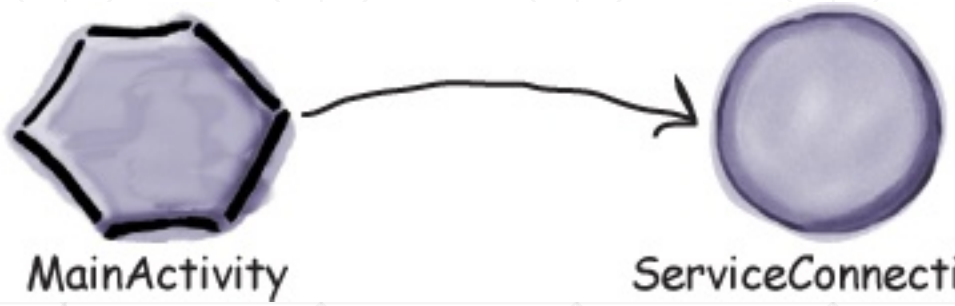

New Android Component
Configure Component
Android Studio
Creates a new service component and adds it to your Android manifest.
Class Name:
OdometerServic
Exported
Enabled
Whether or not components of other applications can invoke the service or interact with it
Cancel
Previous
Next
Finish

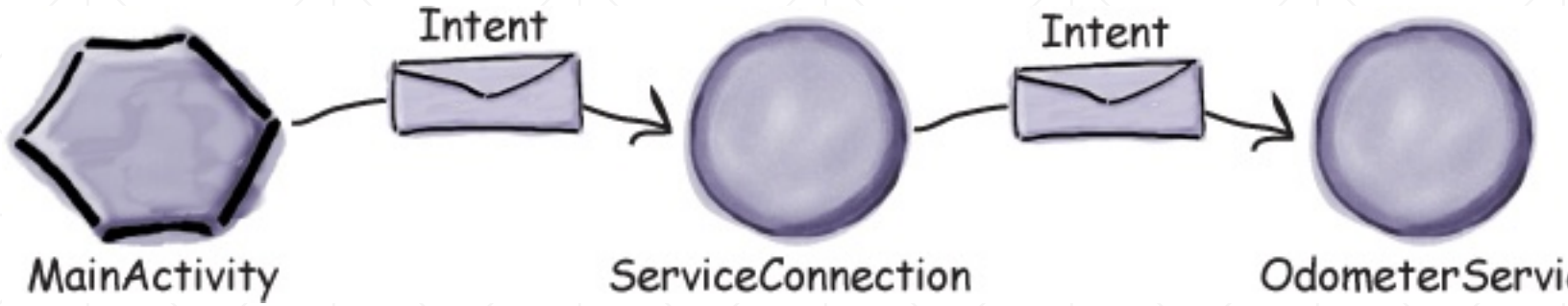
```
package com.hfad.odometer;

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

public class OdometerService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        //Code to bind the service
    }
}
```

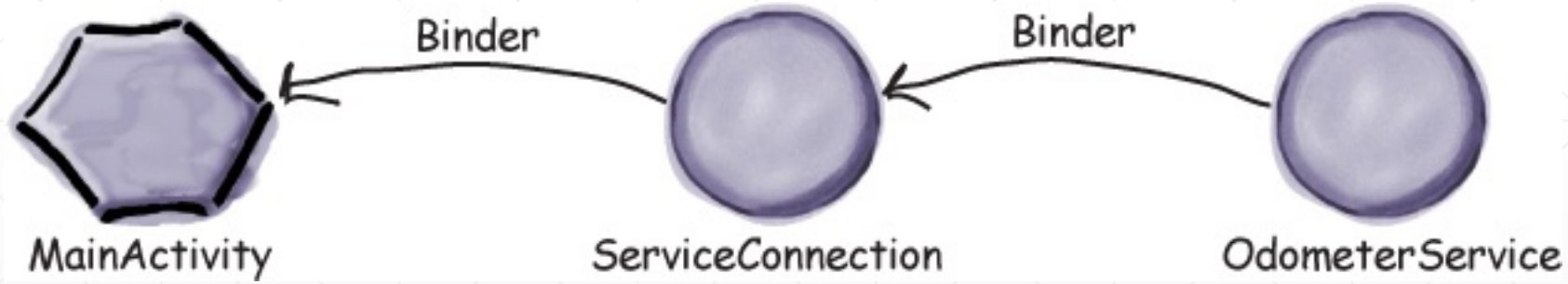
The class extends the Service class.

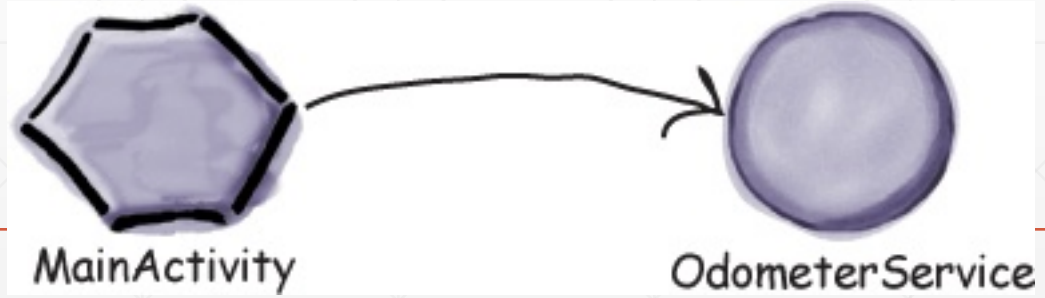
The onBind() method is used for binding components to the service.




MainActivity
ServiceConnection


Intent
Intent
MainActivity
ServiceConnection
OdometerService


Binder
Binder
MainActivity
ServiceConnection
OdometerService


MainActivity
OdometerService

```
public class OdometerBinder extends Binder {
    OdometerService getOdometer() {
        return OdometerService.this;
    }
}
```

When you create a bound service, you need to provide a Binder implementation.

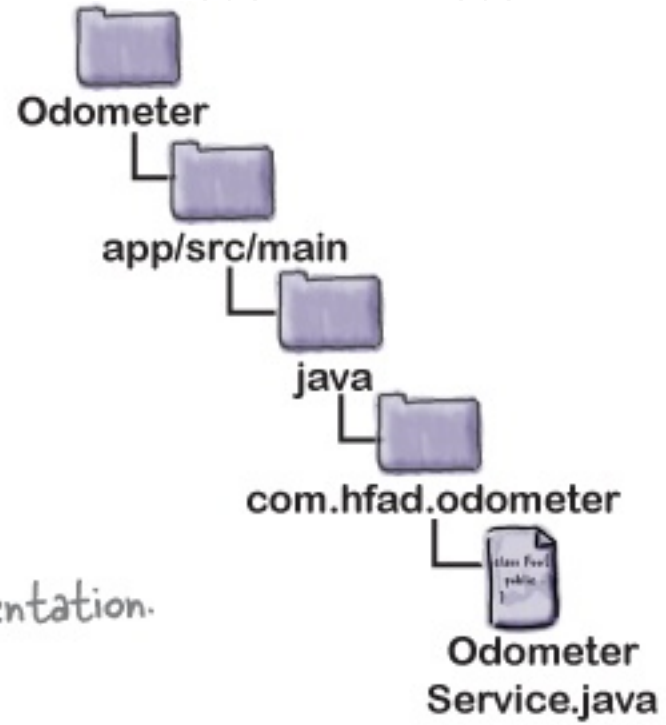
The activity will use this method to get a reference to the OdometerService.

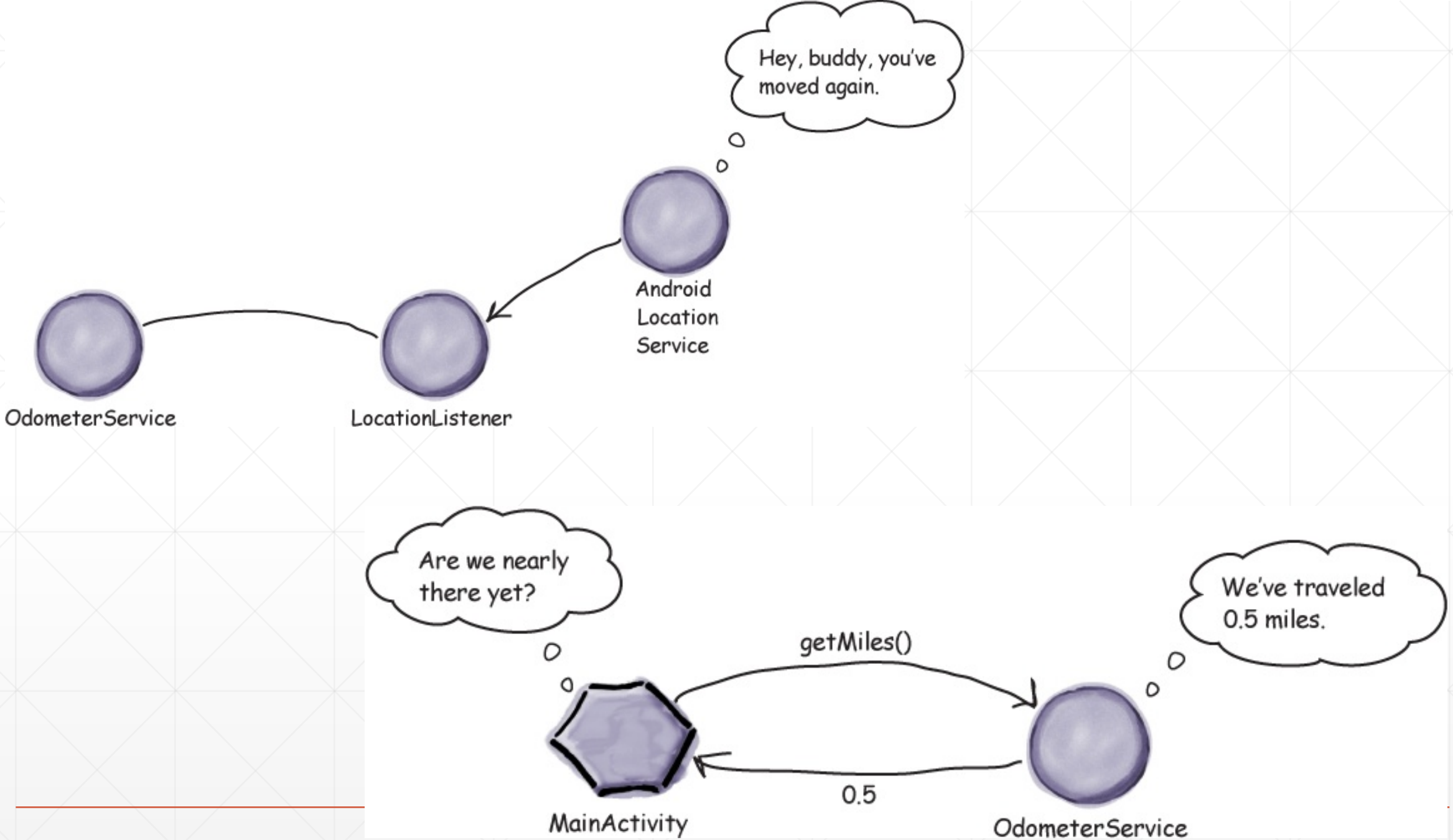
```
...
import android.os.Binder;
import android.os.IBinder;

public class OdometerService extends Service {
    private final IBinder binder = new OdometerBinder();

    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }
...
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
}
```

We're using these classes.

The Binder implementation.

The onBind() method returns an IBinder. This is an interface the Binder class implements.




Hey, buddy, you've moved again.
Android Location Service
OdometerService
LocationListener
Are we nearly there yet?
getMiles()
We've traveled 0.5 miles.
0.5
MainActivity
OdometerService

Method	When it's called	What you use it for
onCreate()	When the service is first created	One-time setup procedures, such as instantiation
onStartCommand()	When an activity starts the service using the startService() method	You don't need to implement this method if your service isn't a started service; it will only run if the service is started using startService()
onBind()	When an activity wants to bind to the service	You must always implement this method by returning an IBinder object; if you don't want activities to bind to the service, return null instead
onDestroy()	When the service is no longer being used and is about to be destroyed	Use this method to clean up any resources

```
@Override
public void onCreate() {
    //Code to set up the listener
}
```

This is what the Service onCreate() method looks like.

This is the new LocationListener.

```
LocationListener listener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        //Code to keep track of the distance
    }

    @Override
    public void onProviderDisabled(String arg0) {}

    @Override
    public void onProviderEnabled(String arg0) {}

    @Override
    public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
};
```

This method gets called whenever the LocationListener is told the device location has changed. The Location parameter describes the current location.

You need to overide these methods too, but they can be left empty. They get called when the GPS is enabled or disabled, or if its status has changed. We don't need to react to any of these events.

```
double distanceInMeters = location.distanceTo(lastLocation);
```

```
...
public class OdometerService extends Service {

    private static double distanceInMeters;
    private static Location lastLocation = null;
    ...

    @Override
    public void onCreate() {
        LocationListener listener = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                if (lastLocation == null) {
                    lastLocation = location;
                }
                distanceInMeters += location.distanceTo(lastLocation);
                lastLocation = location;
            }

            @Override
            public void onProviderDisabled(String arg0) {}

            @Override
            public void onProviderEnabled(String arg0) {}

            @Override
            public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
        };
    }
}
```

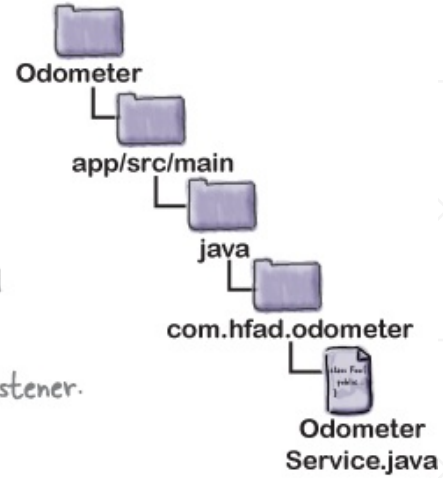
We're storing the distance traveled in meters and the last location as static private variables.

Create the listener.

If it's our first location, set lastLocation to the current Location.

Add the distance between this location and the last to the distanceInMeters variable, and set lastLocation to the current Location.

We need to override these methods, as they're part of the LocationListener interface.



This is how you access the Android location service.

```
LocationManager locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

We used the getSystemService() method earlier to get access to Android's notification service.

This is the GPS provider.

```
locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                                  1000,
                                  1,
                                  listener);
```

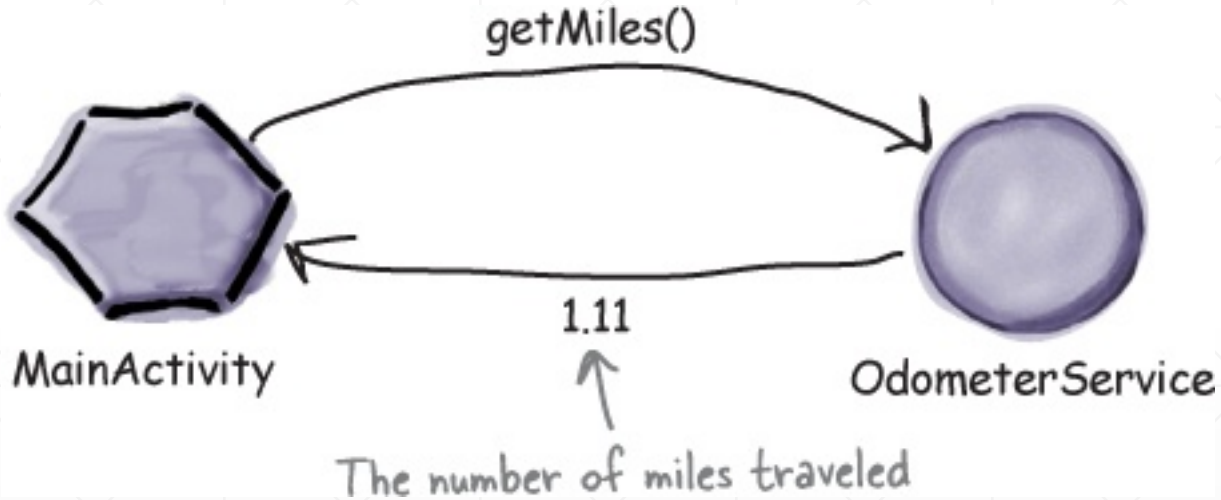
The time in milliseconds.

The distance in meters.

This is the LocationListener we created.

We want to set up the listener and register it with the location service when the service is created.

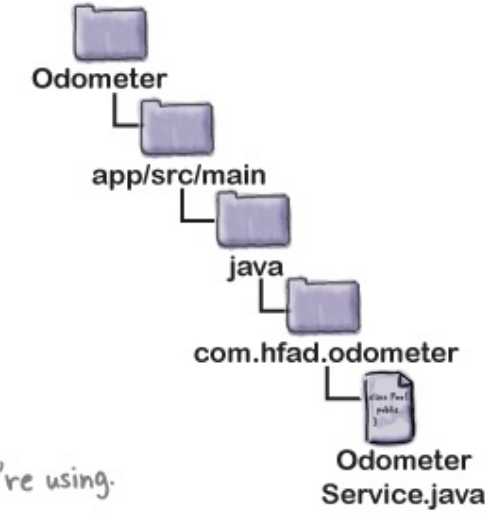
```
@Override
public void onCreate() {
    LocationListener listener = new LocationListener() {...};
    LocationManager locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, listener);
    }
```


getMiles()
1.11
MainActivity
OdometerService
The number of miles traveled

```
public double getMiles() {
    return this.distanceInMeters / 1609.344;
}
```

This converts the distance traveled in meters into miles. We could make this calculation more precise if we wanted to, but it's accurate enough for our purposes.

```
package com.hfad.odometer;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Binder;
import android.os.Bundle;
import android.os.IBinder;

public class OdometerService extends Service {

    private final IBinder binder = new OdometerBinder();
    private static double distanceInMeters;
    private static Location lastLocation = null;

    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }

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



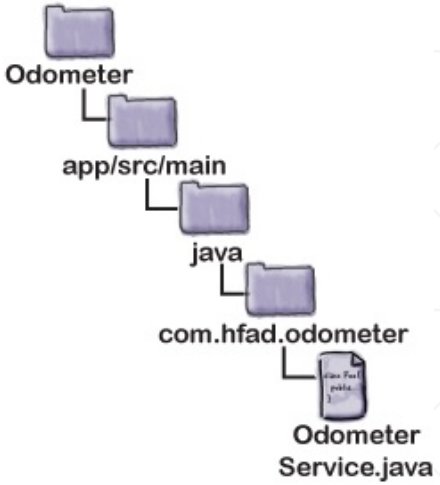
These are all the classes we're using.

These are the private variables we're using.

When you create a bound service, you have to define a Binder object. It enables the activity to bind to the service.

This gets called when the activity binds to the service.

Set up the location listener when the service is created.




```
@Override
public void onCreate() {
    LocationListener listener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            if (lastLocation == null) {
                lastLocation = location;
            }
            distanceInMeters += location.distanceTo(lastLocation);
            lastLocation = location;
        }

        @Override
        public void onProviderDisabled(String arg0) {}

        @Override
        public void onProviderEnabled(String arg0) {}

        @Override
        public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
    };
    LocationManager locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, listener);
}

public double getMiles() {
    return this.distanceInMeters / 1609.344;
}
}
```

This is our implementaton of the location listener.

Convert the distance traveled to miles and return it.

Register the location listener with the location service.

We're adding this because we're using the device GPS in our app.

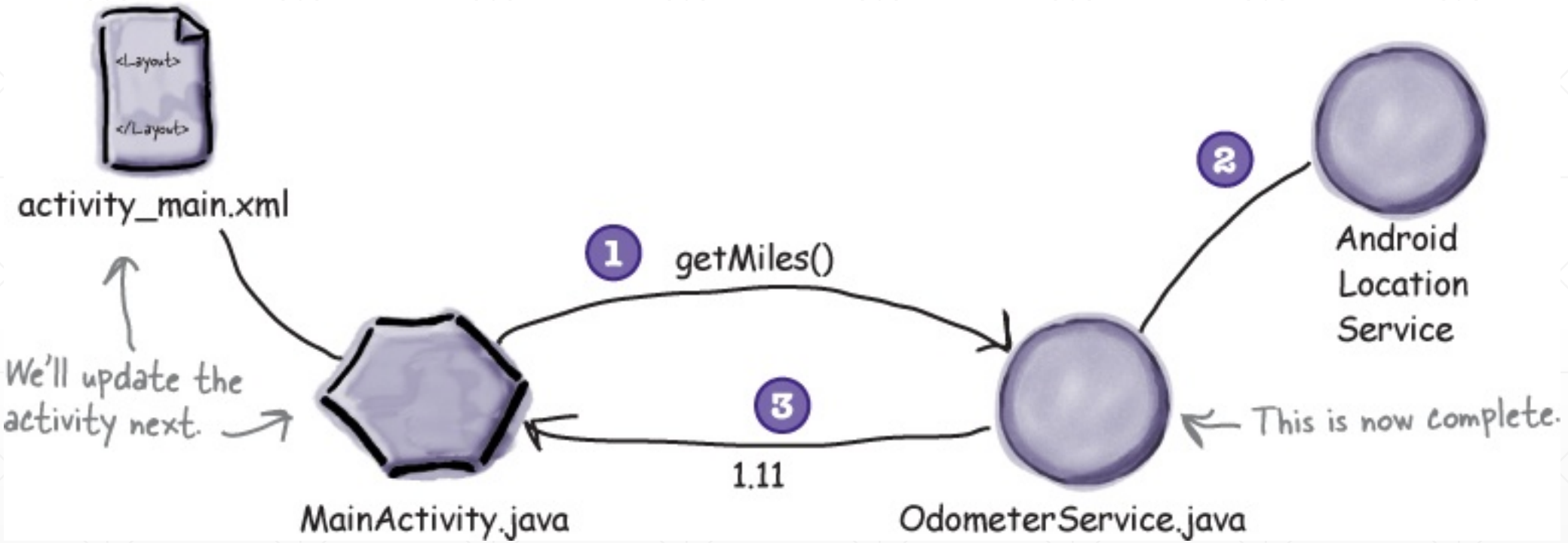
```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
</manifest>
```

```
<manifest ... >

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        ... >
        <activity
            ...
        </activity>

        <service
            android:name=".OdometerService"
            android:exported="false"
            android:enabled="true" >
        </service>
    </application>
</manifest>
```

All services need to be declared in AndroidManifest.xml.

We're setting this to false, as only this app will use the service.

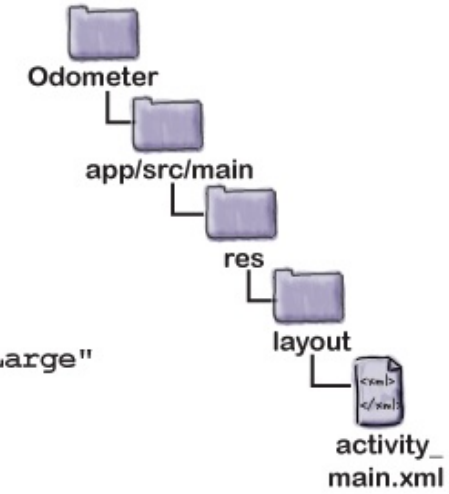
The android:enabled attribute must either be set to true or omitted completely. If you set it to false, your app won't be able to use the service.


<Layout>
</Layout>
activity_main.xml
We'll update the activity next.
1
getMiles()
2
Android Location Service
3
1.11
This is now complete.
MainActivity.java
OdometerService.java

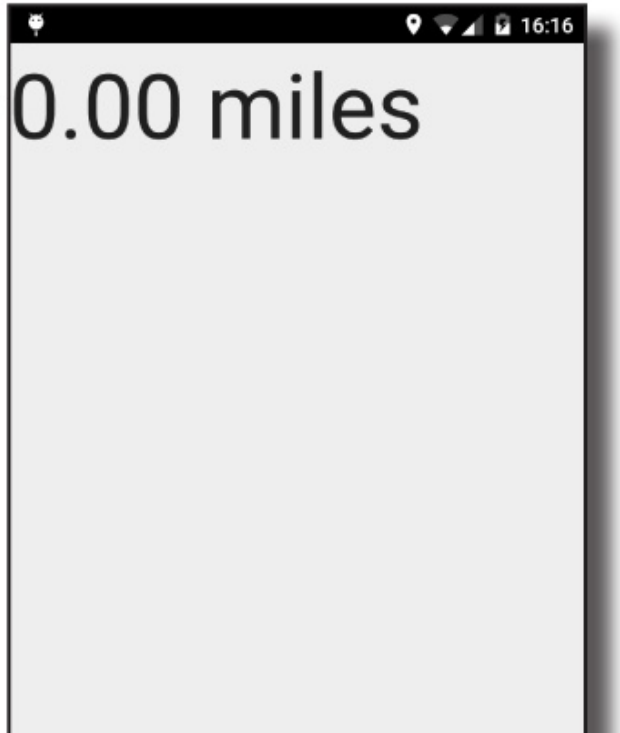
Update MainActivity

- Bind to service
- Display miles

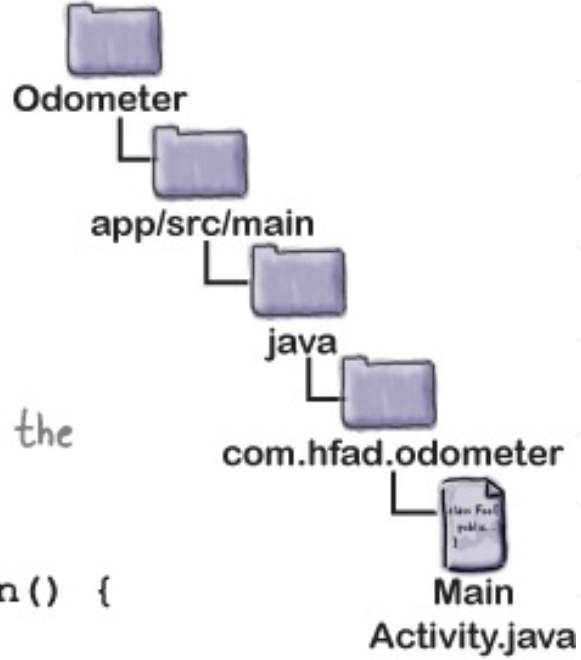
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView android:text=""
        android:id="@+id/distance"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_centerHorizontal="true"
        android:singleLine="false"
        android:textSize="60dp"/>
</RelativeLayout>
```



We'll use the TextView to display the distance.




Main
Activity
Service
Connection
Odometer
Service




```
...
public class MainActivity extends Activity {

    private OdometerService odometer;
    private boolean bound = false;
    ...

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder binder) {
            OdometerService.OdometerBinder odometerBinder =
                                    (OdometerService.OdometerBinder) binder;
            odometer = odometerBinder.getOdometer();
            bound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            bound = false;
        }
    };
}
```

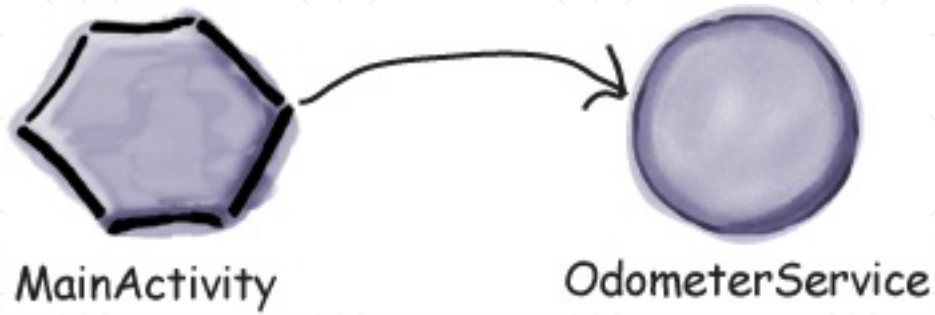
We'll use this for the OdometerService.

Use this to store whether or not the activity's bound to the service.

Cast the Binder to an OdometerBinder, then use to get a reference to the OdometerService.

When the service is connected, set bound to true.

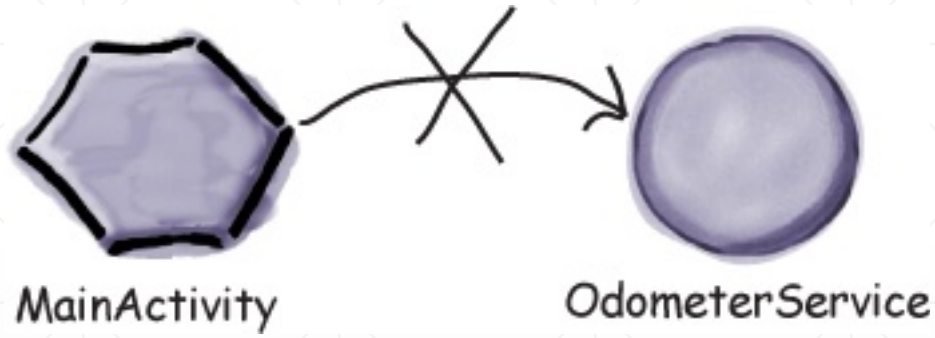
When the service is disconnected, set bound to false.


MainActivity
OdometerService

```
@Override
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, OdometerService.class);
    bindService(intent, connection, Context.BIND_AUTO_CREATE);
}
```

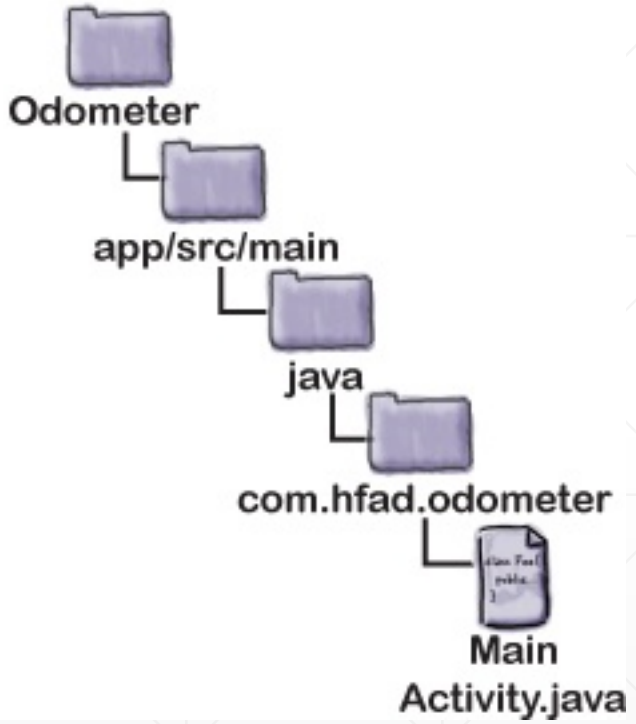
This is an intent directed to the OdometerService.

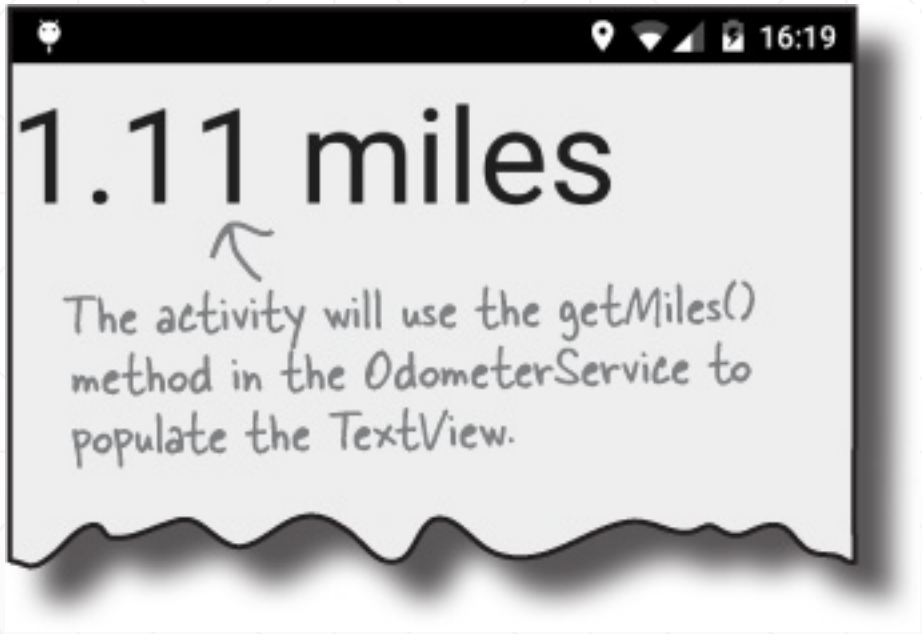
This uses the intent and service connection to bind the activity to the service.


MainActivity
OdometerService

```
@Override
protected void onStop() {
    super.onStop();
    if (bound) {
        unbindService(connection);
        bound = false;
    }
}
```

This uses the service connection to unbind from the service.




16:19
1.11 miles
The activity will use the getMiles() method in the OdometerService to populate the TextView.

```
private void watchMileage() {
    final TextView distanceView = (TextView)findViewById(R.id.distance);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            double distance = 0.0;
            if (odometer != null) {
                distance = odometer.getMiles();
            }
            String distanceStr = String.format("%1$,.2f miles", distance);
            distanceView.setText(distanceStr);
            handler.postDelayed(this, 1000);
        }
    });
}
```

Get the text view.

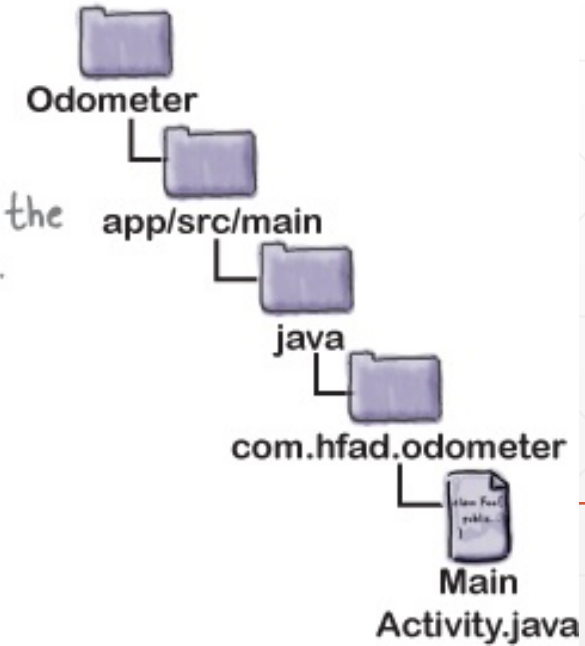
Create a new Handler.

Call the post() method, passing in a new Runnable.

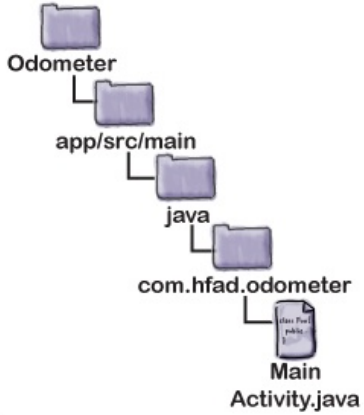
If we've got a reference to the OdometerService, use its getMiles() method.

Format the miles.

Post the code in the Runnable to be run again after a delay of 1,000 milliseconds, or 1 second. As this line of code is included in the Runnable run() method, it will run every second (with a slight lag).



```
package com.hfad.odometer;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.widget.TextView;

public class MainActivity extends Activity {

    private OdometerService odometer;
    private boolean bound = false;

    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder binder) {
            OdometerService.OdometerBinder odometerBinder =
                                    (OdometerService.OdometerBinder) binder;
            odometer = odometerBinder.getOdometer();
            bound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            bound = false;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        watchMileage();
    }
```



We'll use this for the OdometerService.

Use this to store whether or not the activity's bound to the service.

We need to define a ServiceConnection.

Get a reference to the OdometerService when the service is connected.

Call the watchMileage() function when the activity's created.

```
@Override
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, OdometerService.class);
    bindService(intent, connection, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop() {
    super.onStop();
    if (bound) {
        unbindService(connection);
        bound = false;
    }
}

private void watchMileage() {
    final TextView distanceView = (TextView)findViewById(R.id.distance);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            double distance = 0.0;
            if (odometer != null) {
                distance = odometer.getMiles();
            }
            String distanceStr = String.format("%1$,.2f miles", distance);
            distanceView.setText(distanceStr);
            handler.postDelayed(this, 1000);
        }
    });
}
}
```

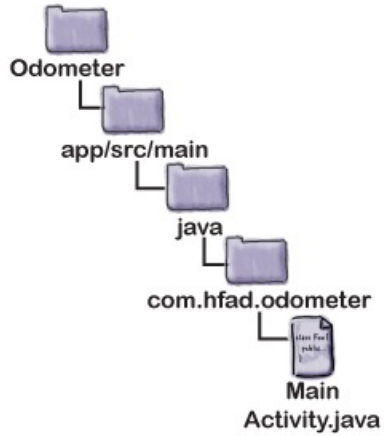
Bind the service when the activity starts.

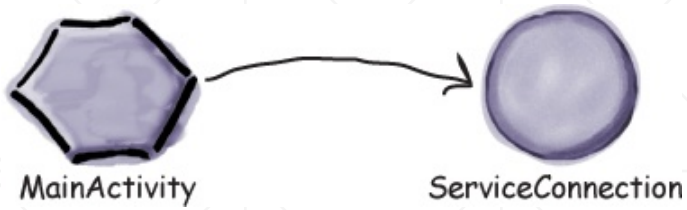
Unbind the service when the activity stops.

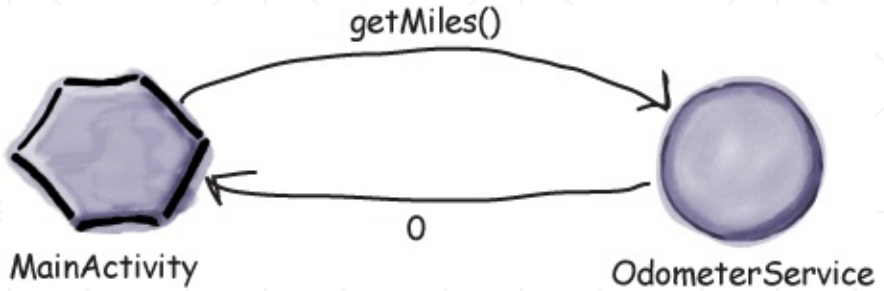
This method updates the mileage that's displayed.

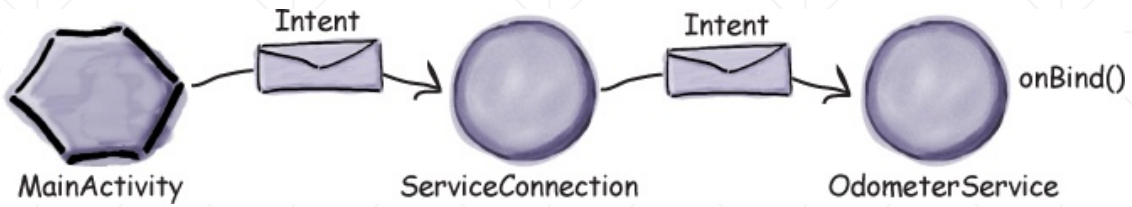
If we've got a reference to the OdometerService, use its getMiles() method.

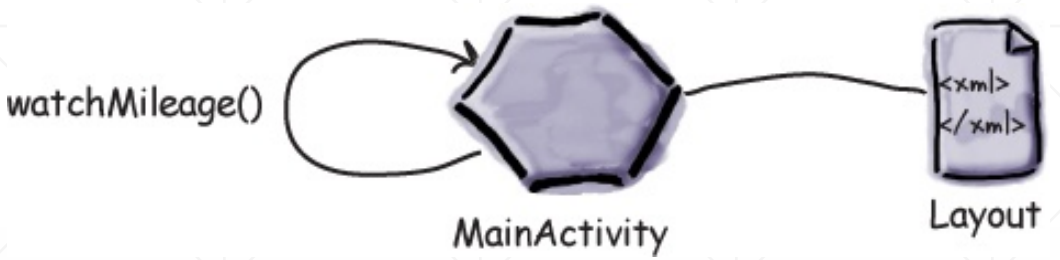
Update the distance every second.

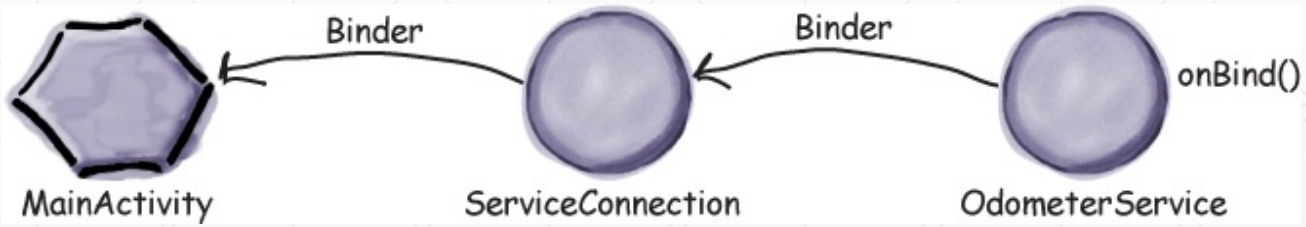


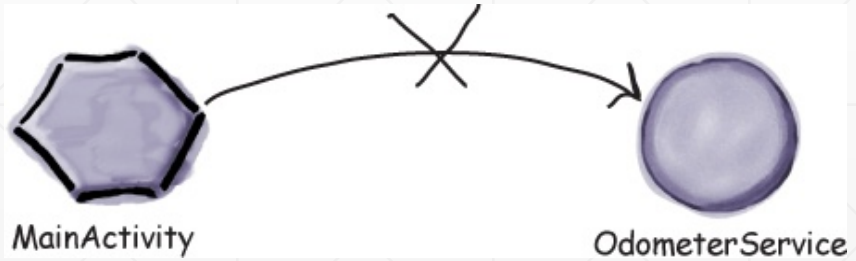

MainActivity
ServiceConnection

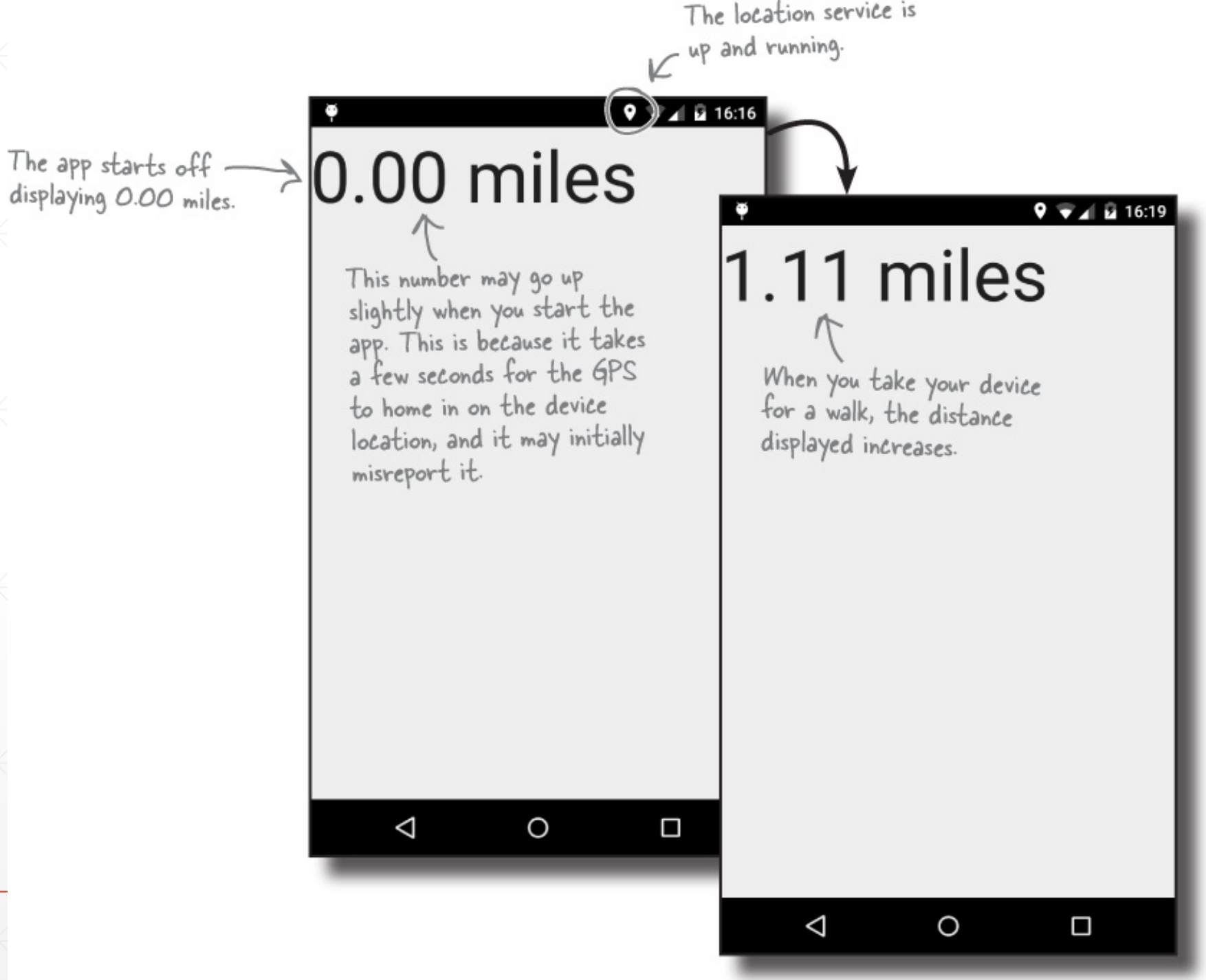

getMiles()
0
MainActivity
OdometerService


Intent
Intent
onBind()
MainActivity
ServiceConnection
OdometerService


watchMileage()
<xml>
</xml>
MainActivity
Layout


Binder
Binder
onBind()
MainActivity
ServiceConnection
OdometerService


MainActivity
OdometerService


The location service is up and running.
The app starts off displaying 0.00 miles.
16:16
0.00 miles
This number may go up slightly when you start the app. This is because it takes a few seconds for the GPS to home in on the device location, and it may initially misreport it.
16:19
1.11 miles
When you take your device for a walk, the distance displayed increases.

Bullet Points

- A service is a component that can perform tasks in the background. It doesn't have a user interface.
- A started service can run in the background indefinitely, even when the activity that started it is destroyed. Once the operation is done, it stops itself.
- You declare services in *AndroidManifest.xml* using the <service> element.
- You can create a simple started service by extending the IntentService class and overriding its onHandleIntent() method. The IntentServiceclass is designed for handling intents.
- You start a started service using the startService() method.
- If you override the IntentService onStartCommand() method, you must call its super implementation.
- You create a notification using a notification builder. You get your notification to start an activity using a pending intent. You then use Android's notification service to display the notification.
- A bound service is bound to another component such as an activity. The activity can interact with it and get results.

Bullet Points

- You usually create a bound service by extending the Service class. You must define your own Binder object, and override the onBind() method. This is called when a component wants to bind to the service.
- The Service onCreate() method is called when the service is created. Use it for instantiation.
- The Service onDestroy() method is called when the service is about to be destroyed.
- You can use the Android location service to get the current location of the device. You create a LocationListener, and then register it with the location service. You can add criteria for how often the listener is notified of changes. When you use the device GPS, you need to add a permission for it in*AndroidManifest.xml*.
- To bind an activity to a service, you create a ServiceConnection. You override the onServiceConnected() method to get a reference to the service.
- You bind to the service using the bindService() method. You unbind from the service using the unbindService() method.

Network Access

- Choosing HTTP Client
  - Apache HTTP client
  - Java’s HttpURLConnection
  - Volley
  - Retrofit

HTTPUrlConnection

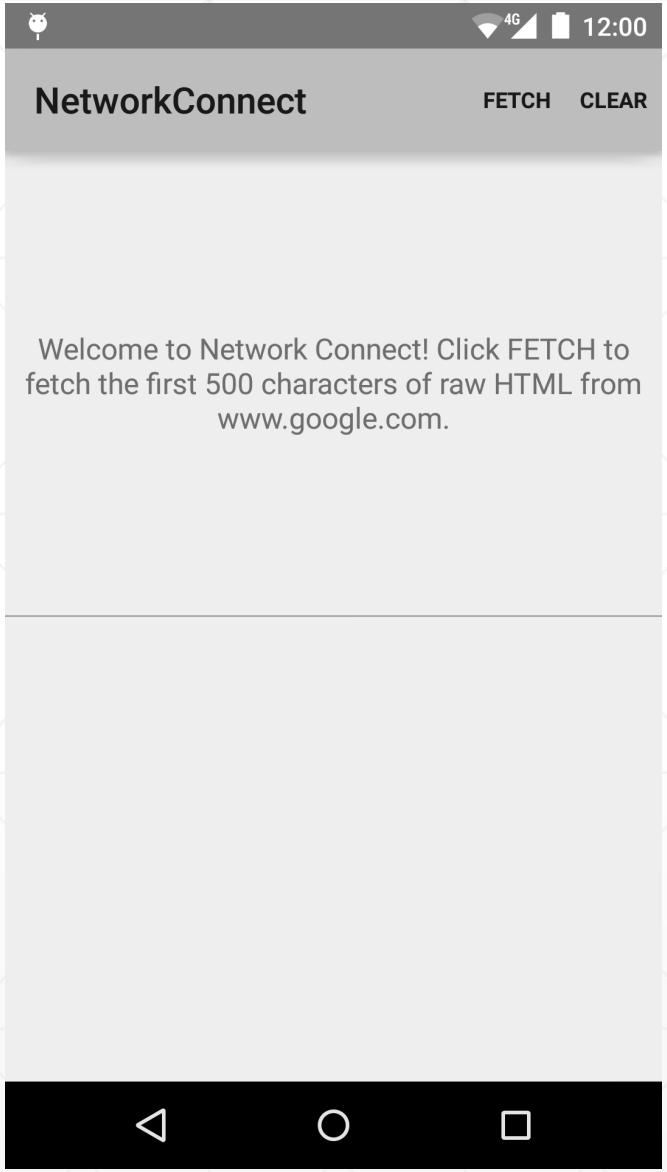
```
URL url = new URL("http://www.android.com/");

HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

InputStream in = new BufferedInputStream(urlConnection.getInputStream());

readStream(in);
```

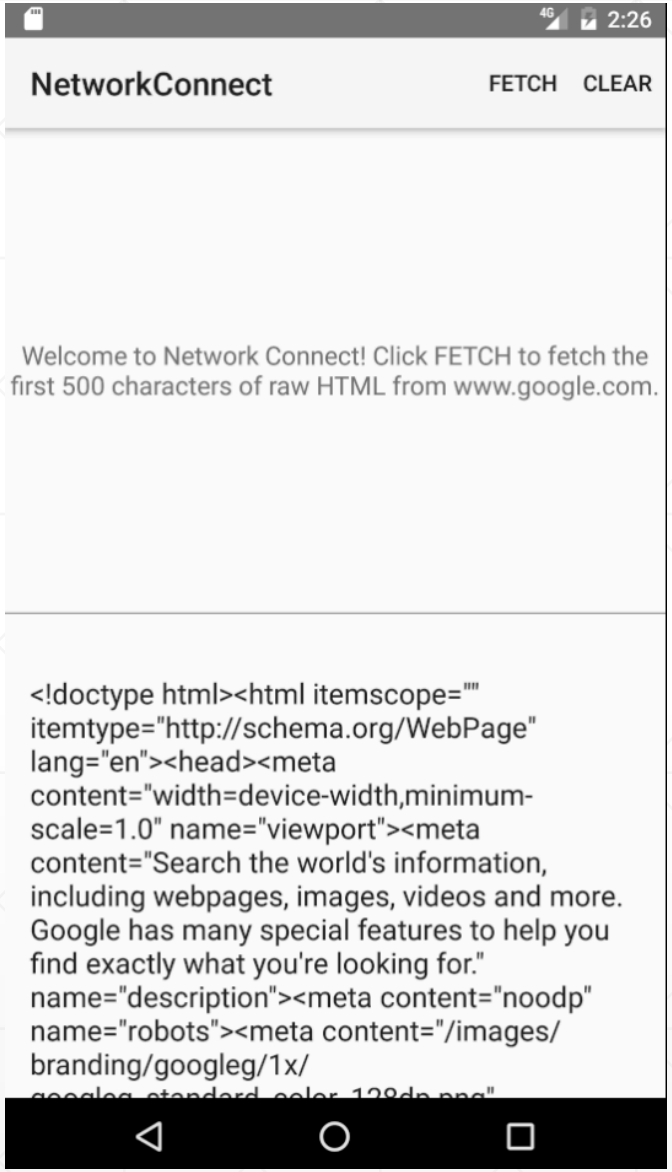
HTTPUrlConnection

```
URL url = new URL(urlString);
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setReadTimeout(10000 /* milliseconds */);
conn.setConnectTimeout(15000 /* milliseconds */);
conn.setRequestMethod("GET");
conn.setDoInput(true);
// Start the query
conn.connect();
InputStream stream = conn.getInputStream();
return stream;
```

Network Access

- Google Example
  - https://github.com/googlesamples/android-NetworkConnect
- Fetch raw HTML using HttpURLConnection
- AsyncTask used to fetch on background task.


12:00
NetworkConnect
FETCH
CLEAR
Welcome to Network Connect! Click FETCH to fetch the first 500 characters of raw HTML from www.google.com.


2:26
NetworkConnect
FETCH
CLEAR
Welcome to Network Connect! Click FETCH to fetch the first 500 characters of raw HTML from www.google.com.
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="width=device-width,minimum-scale=1.0" name="viewport"><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking for." name="description"><meta content="noodp" name="robots"><meta content="/images/branding/googleg/1x/

Network Access

- Create AyncTask
- Download URL
- HTTP Connection
- Read the stream

```
private class MyAsyncTask extends AsyncTask<Params, Progress, Result>

        protected void onPreExecute() {
                //Code to run before executing the task
         }

        protected Result doInBackground(Params... params) {
                //Code that you want to run in a background thread
        }

        protected void onProgressUpdate(Progress... values) {
                //Code that you want to run to publish the progress of your task
        }

        protected void onPostExecute(Result result) {
                //Code that you want to run when the task is complete
        }
}
```

```
/**
 * Implementation of AsyncTask, to fetch the data in the background away from
 * the UI thread.
 */
private class DownloadTask extends AsyncTask<String, Void, String> {

    @Override
    protected String doInBackground(String... urls) {
        try {
            return loadFromNetwork(urls[0]);
        } catch (IOException e) {
          return getString(R.string.connection_error);
        }
    }

    /**
     * Uses the logging framework to display the output of the fetch
     * operation in the log fragment.
     */
    @Override
    protected void onPostExecute(String result) {
      Log.i(TAG, result);
    }
}
```

```java
/** Initiates the fetch operation. */
private String loadFromNetwork(String urlString) throws IOException {
    InputStream stream = null;
    String str ="";

    try {
        stream = downloadUrl(urlString);
        str = readIt(stream, 500);
   } finally {
       if (stream != null) {
           stream.close();
       }
    }
    return str;
}
```

```java
/**
 * Given a string representation of a URL, sets up a connection and gets
 * an input stream.
 * @param urlString A string representation of a URL.
 * @return An InputStream retrieved from a successful HttpURLConnection.
 * @throws java.io.IOException
 */
private InputStream downloadUrl(String urlString) throws IOException {
    // BEGIN_INCLUDE(get_inputstream)
    URL url = new URL(urlString);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setReadTimeout(10000 /* milliseconds */);
    conn.setConnectTimeout(15000 /* milliseconds */);
    conn.setRequestMethod("GET");
    conn.setDoInput(true);
    // Start the query
    conn.connect();
    InputStream stream = conn.getInputStream();
    return stream;
    // END_INCLUDE(get_inputstream)
}
```

```java
/** Reads an InputStream and converts it to a String.
 * @param stream InputStream containing HTML from targeted site.
 * @param len Length of string that this method returns.
 * @return String concatenated according to len parameter.
 * @throws java.io.IOException
 * @throws java.io.UnsupportedEncodingException
 */
private String readIt(InputStream stream, int len) throws IOException, UnsupportedEncodingExc
    Reader reader = null;
    reader = new InputStreamReader(stream, "UTF-8");
    char[] buffer = new char[len];
    reader.read(buffer);
    return new String(buffer);
}
```

```
/**
 * Implementation of AsyncTask, to fetch the data in the background away from
 * the UI thread.
 */
private class DownloadTask extends AsyncTask<String, Void, String> {

    @Override
    protected String doInBackground(String... urls) {
        try {
            return loadFromNetwork(urls[0]);
        } catch (IOException e) {
          return getString(R.string.connection_error);
        }
    }

    /**
     * Uses the logging framework to display the output of the fetch
     * operation in the log fragment.
     */
    @Override
    protected void onPostExecute(String result) {
      Log.i(TAG, result);
    }
}
```