18
Servicios

Los servicios trabajan atrás del escenario

Started services

A started service can run in the background indefinitely, even when the activity that started it is destroyed. If you wanted to download a large file from the Internet, you would probably create it as a started service. Once the operation is done, the service stops.

Bound services

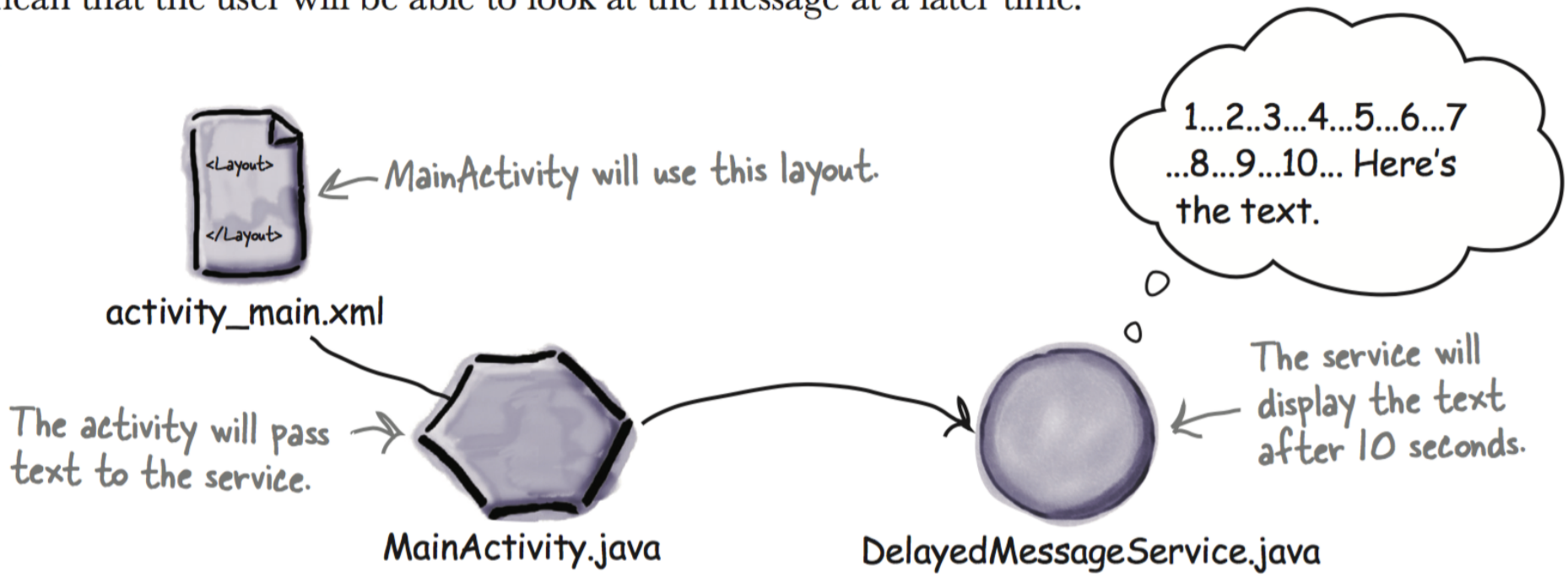
A bound service is bound to another component such as an activity. The activity can interact with it, send requests, and get results. A bound service runs as long as components are bound to it. When components are no longer bound to it, the service is destroyed. If you wanted to create an odometer to measure the distance traveled by a vehicle, you'd probably use a bound service. This way, any activities bound to the service could keep asking the service for updates on the distance traveled.

La aplicación que inicia el servicio

1. **Display the message in the log.**
We'll start by displaying the message in the log so that we can check the service works OK. We can look at the log in Android Studio.

2. **Display the message in a Toast.**
We'll get the message to appear in a pop-up toast so that you don't have to keep your device connected to Android Studio in order to see it working.

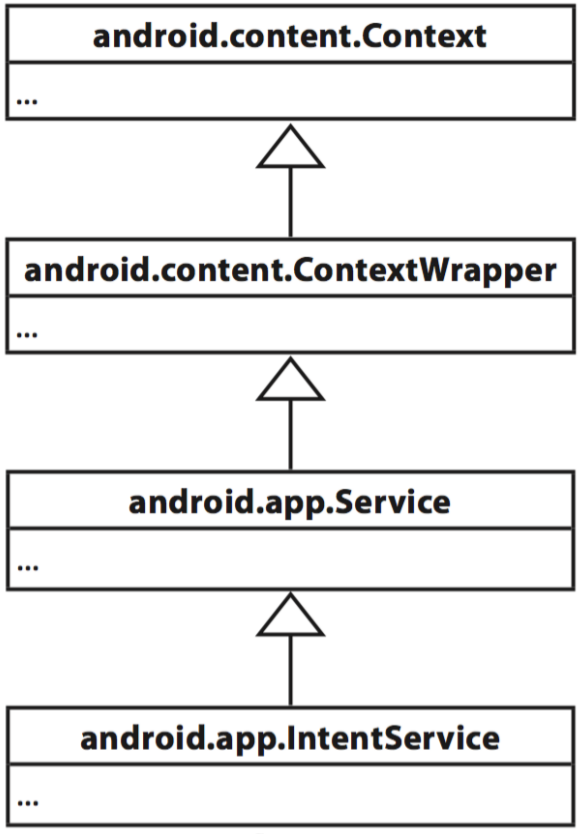
3. **Display the message in a Notification.**
We'll get `DelayedMessageService` to use Android's built-in notification service to display the message in a notification. This will mean that the user will be able to look at the message at a later time.



```
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.

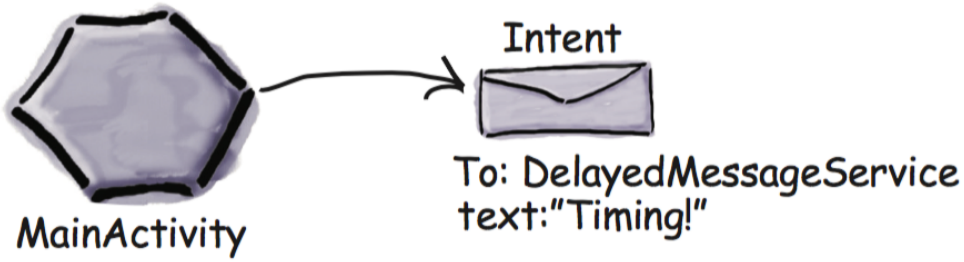


This is the Service class hierarchy.

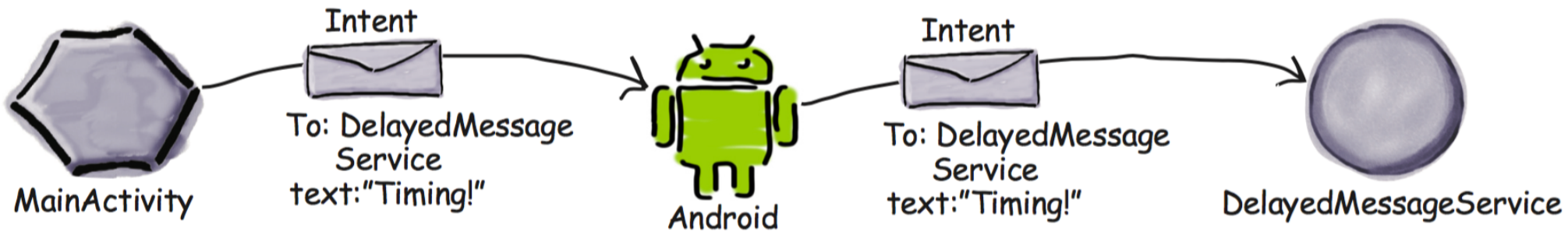
Creando un intent servicio

El IntentService visto desde arriba

1 **An activity says what service it needs to call by creating an explicit intent.**
The intent specifies the service it's intended for.



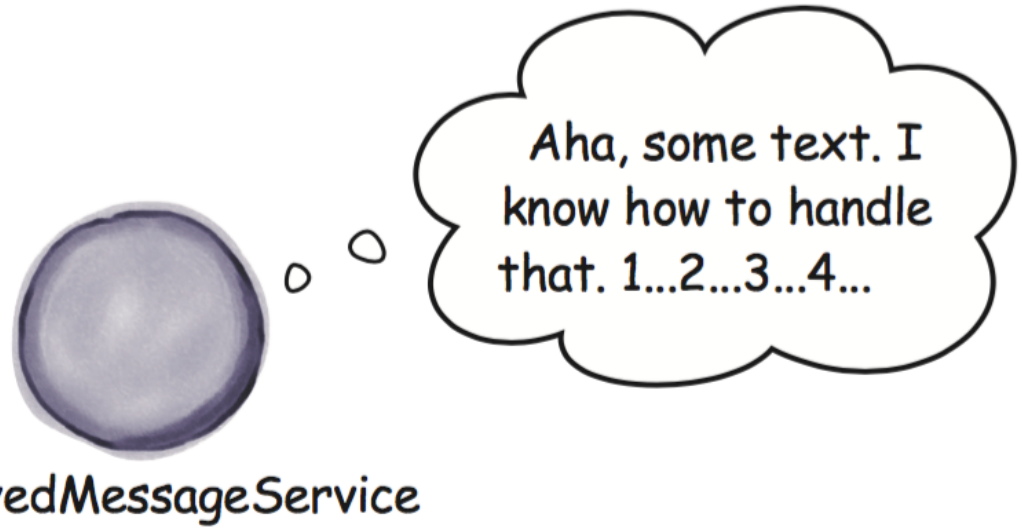
2 **The intent is passed to the service.**



El IntentService visto desde arriba

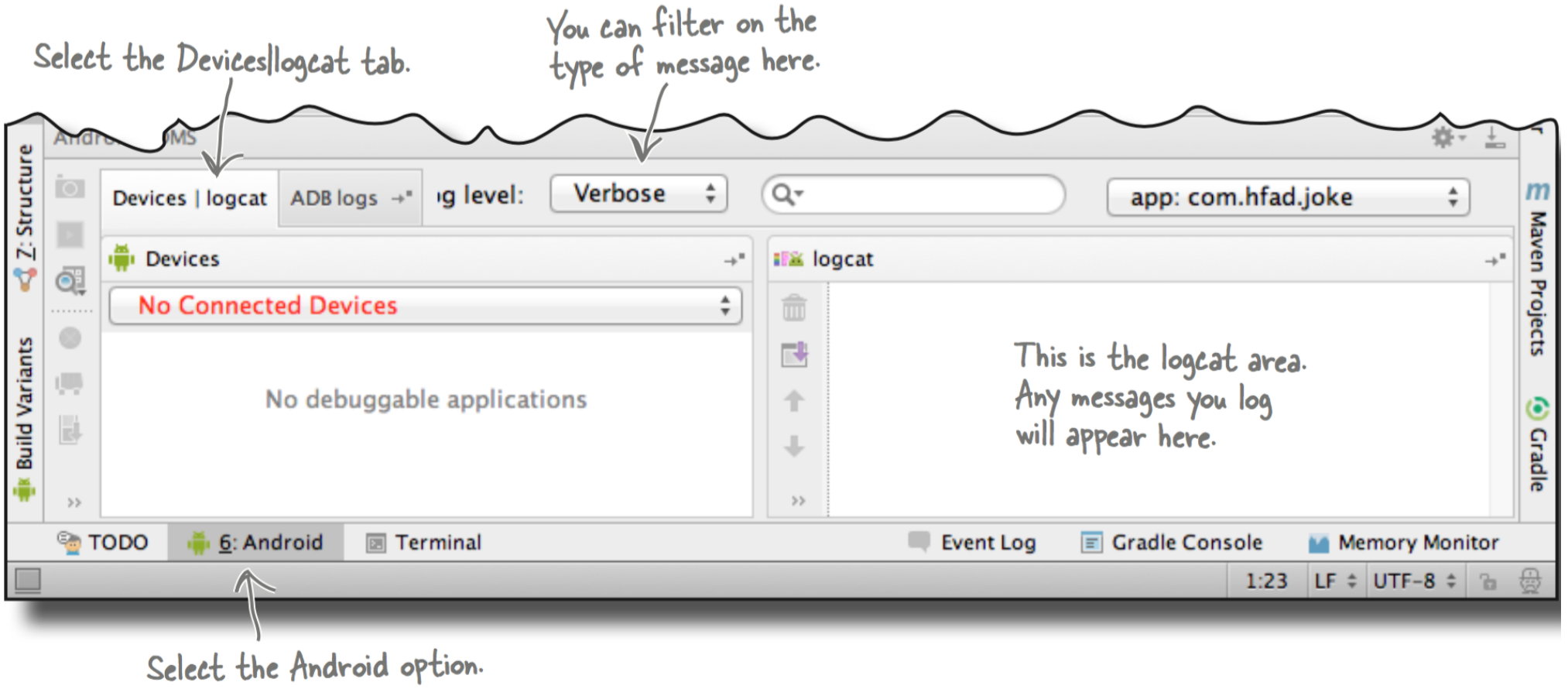
3 **The service starts and handles the intent.**

The `IntentService` `onHandleIntent()` method gets called and runs in a separate thread. If the service is passed multiple intents, it deals with them in sequence, one at a time. Once the service has finished running, it stops.



Como mostrar mensajes

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.



El código completo de DelayedMessageService

```
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";
```

Extend the IntentService class.

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

```
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);
}
}
```

Call the super constructor.

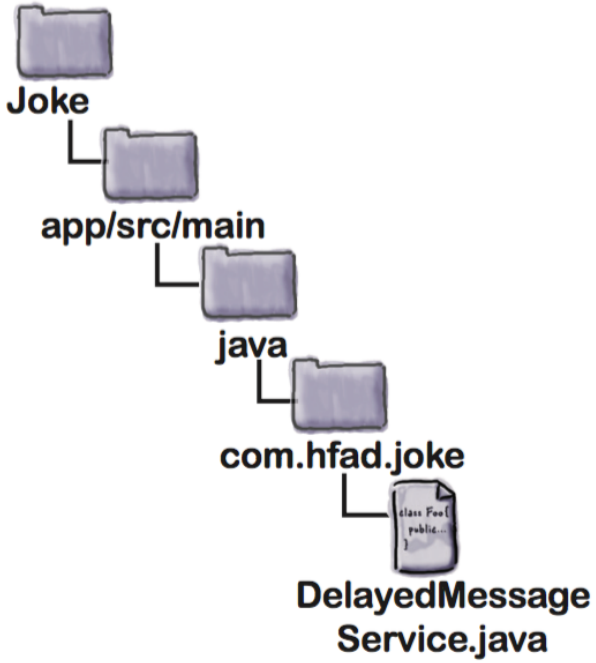
This method contains the code you want to run when the service receives an intent.

Wait 10 seconds.

Get the text from the intent

Call the showText() method.

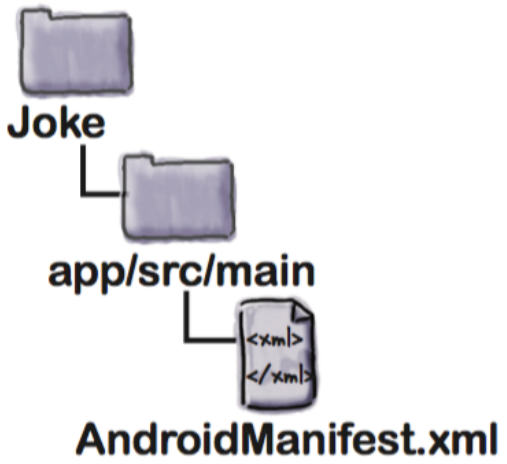
This logs a piece of text so we can see it in the logcat through Android Studio.



El código completo de DelayedMessageService

Declaración de servicios en AndroidManifest.xml

```
<?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.

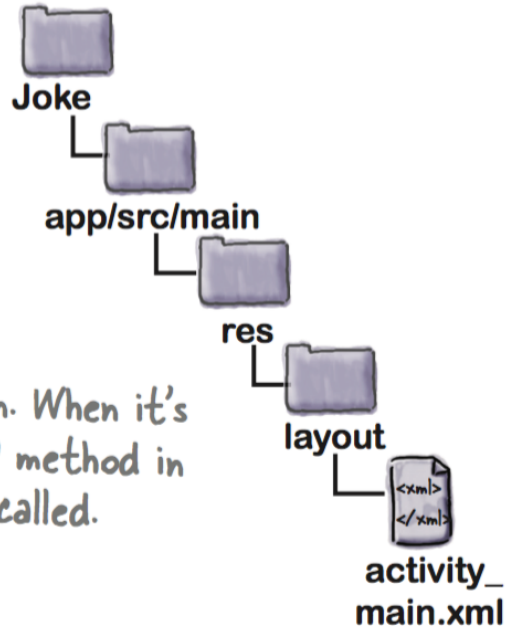
The service name has a . in front of it so that Android can combine it with the package name to derive the fully qualified class name.

```
<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.



Agregando un botón en activity_main.xml

```
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);
    }
}
```

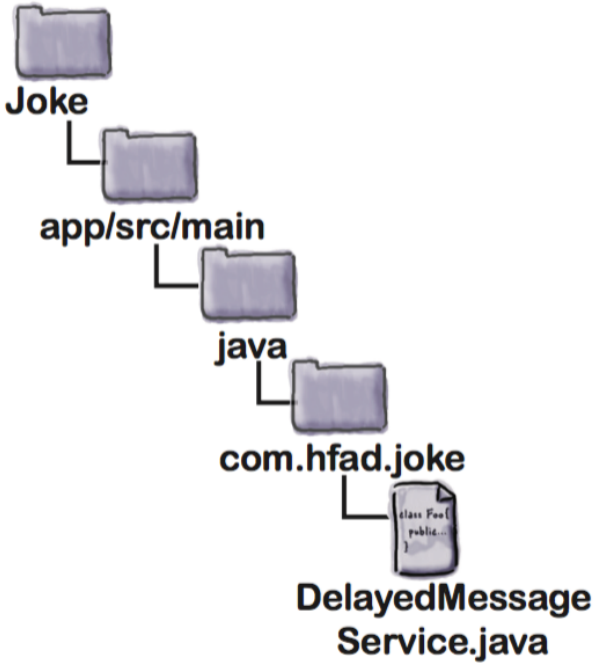
We're using these classes.

This will run when the button gets clicked.

Create the intent.

Add text to the intent.

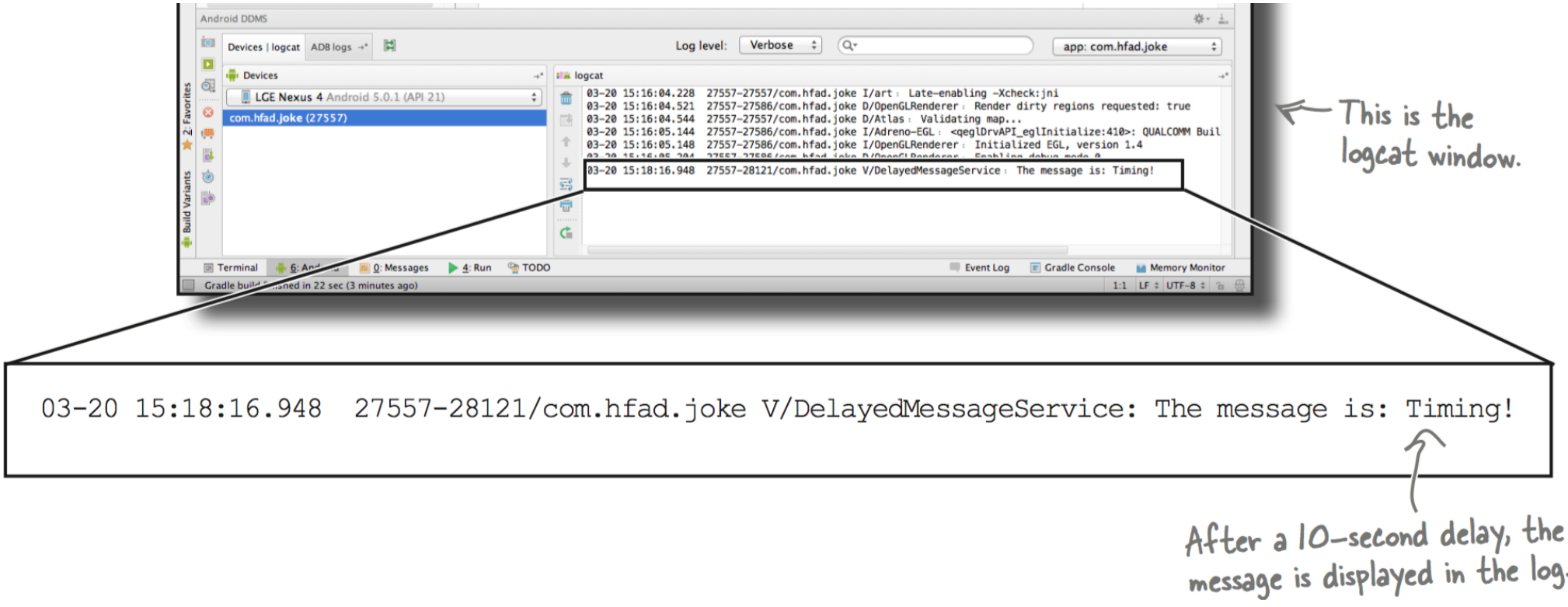
Start the service.



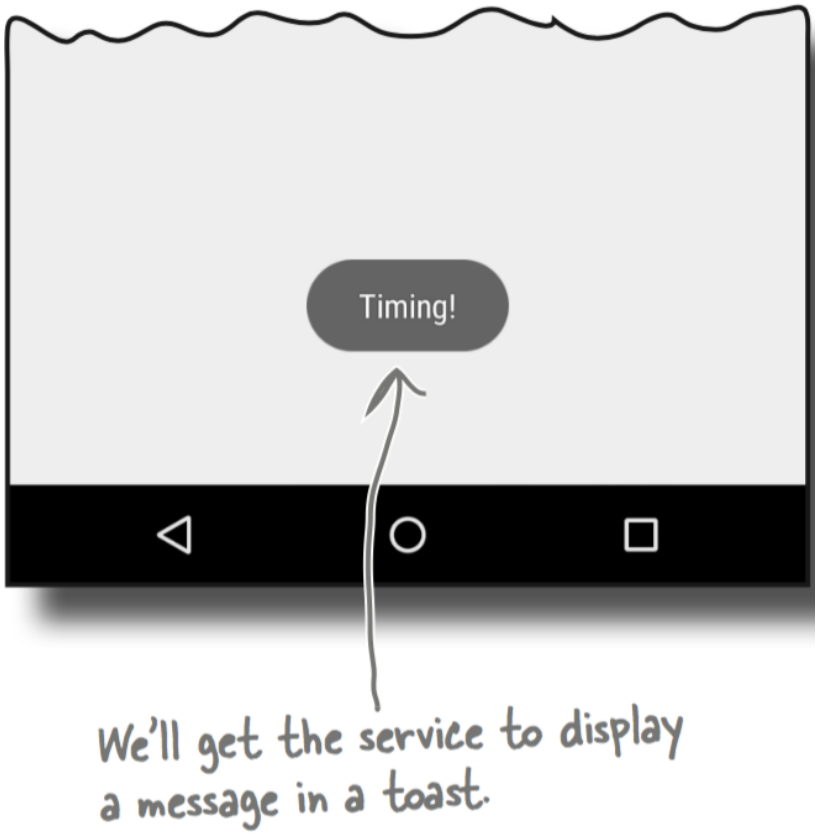
Iniciando un servicio con startService()

Prueba


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


Android DDMS
Devices | logcat
ADB logs
Log level:
Verbose
app: com.hfad.joke
Devices
LGE Nexus 4 Android 5.0.1 (API 21)
com.hfad.joke (27557)
logcat
03-20 15:16:04.228 27557-27557/com.hfad.joke I/art: Late-enabling -Xcheck:jni
03-20 15:16:04.521 27557-27586/com.hfad.joke D/OpenGLRenderer: Render dirty regions requested: true
03-20 15:16:04.544 27557-27557/com.hfad.joke D/Atlas: Validating map...
03-20 15:16:05.144 27557-27586/com.hfad.joke I/Adreno-EGL: <qeglDrvAPI_eglInitialize:410>: QUALCOMM Buil
03-20 15:16:05.148 27557-27586/com.hfad.joke I/OpenGLRenderer: Initialized EGL, version 1.4
03-20 15:18:16.948 27557-28121/com.hfad.joke V/DelayedMessageService: The message is: Timing!
This is the logcat window.
Terminal
0: Messages
4: Run
TODO
Event Log
Gradle Console
Memory Monitor
1:1
LF
UTF-8
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.

Queremos mandar un mensaje a la pantalla



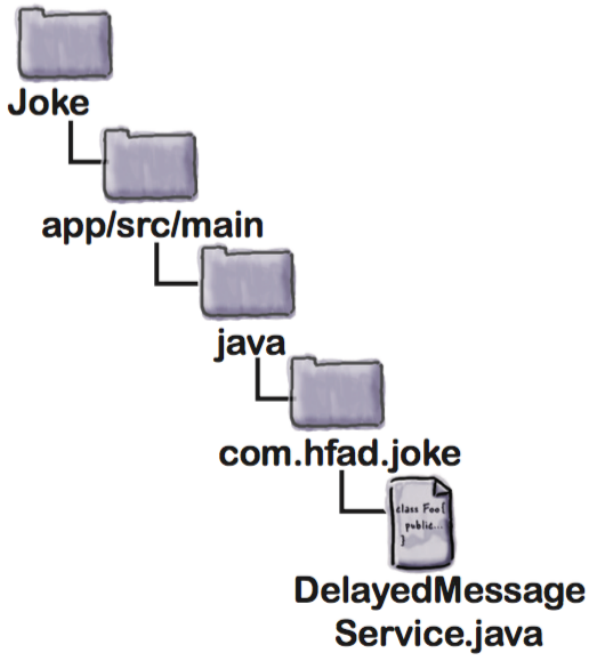
- Create a handler in the main thread.
- Use the `Handler` `post()` method in the service `onHandleIntent()` method to display a toast.

```
...
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.



onStartCommand() se ejecuta en el hilo principal

El código completo de DelayedMessageService

```
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);
    }
```

We're using these extra classes.

Add the handler as a new private variable.

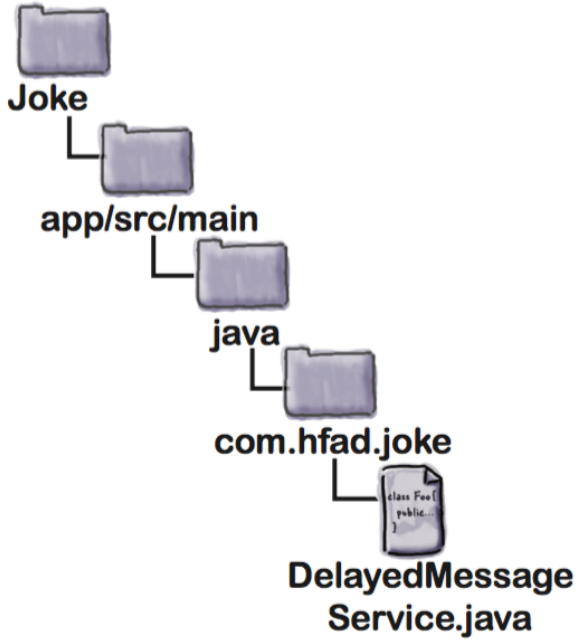
Create the handler on the main thread.

We're not changing this method.

```
    @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();
            }
        });
    }
}
```

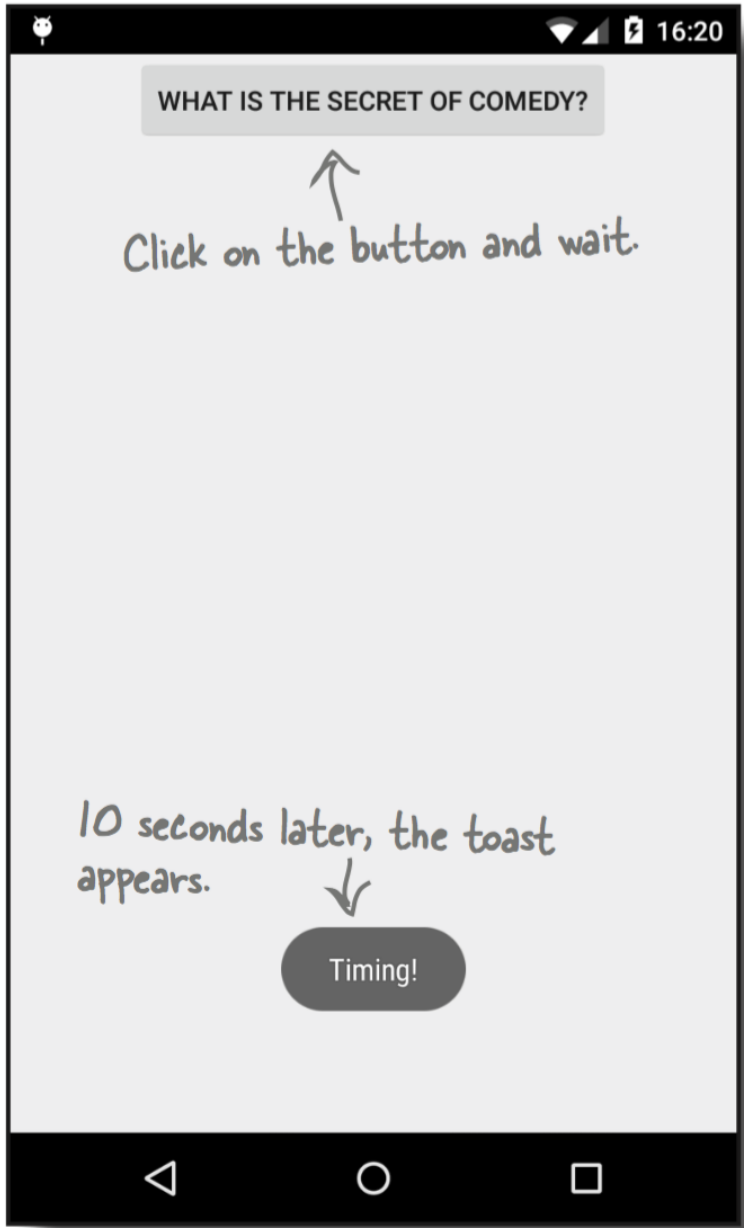
Post the Toast code to the main thread using the handler.

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

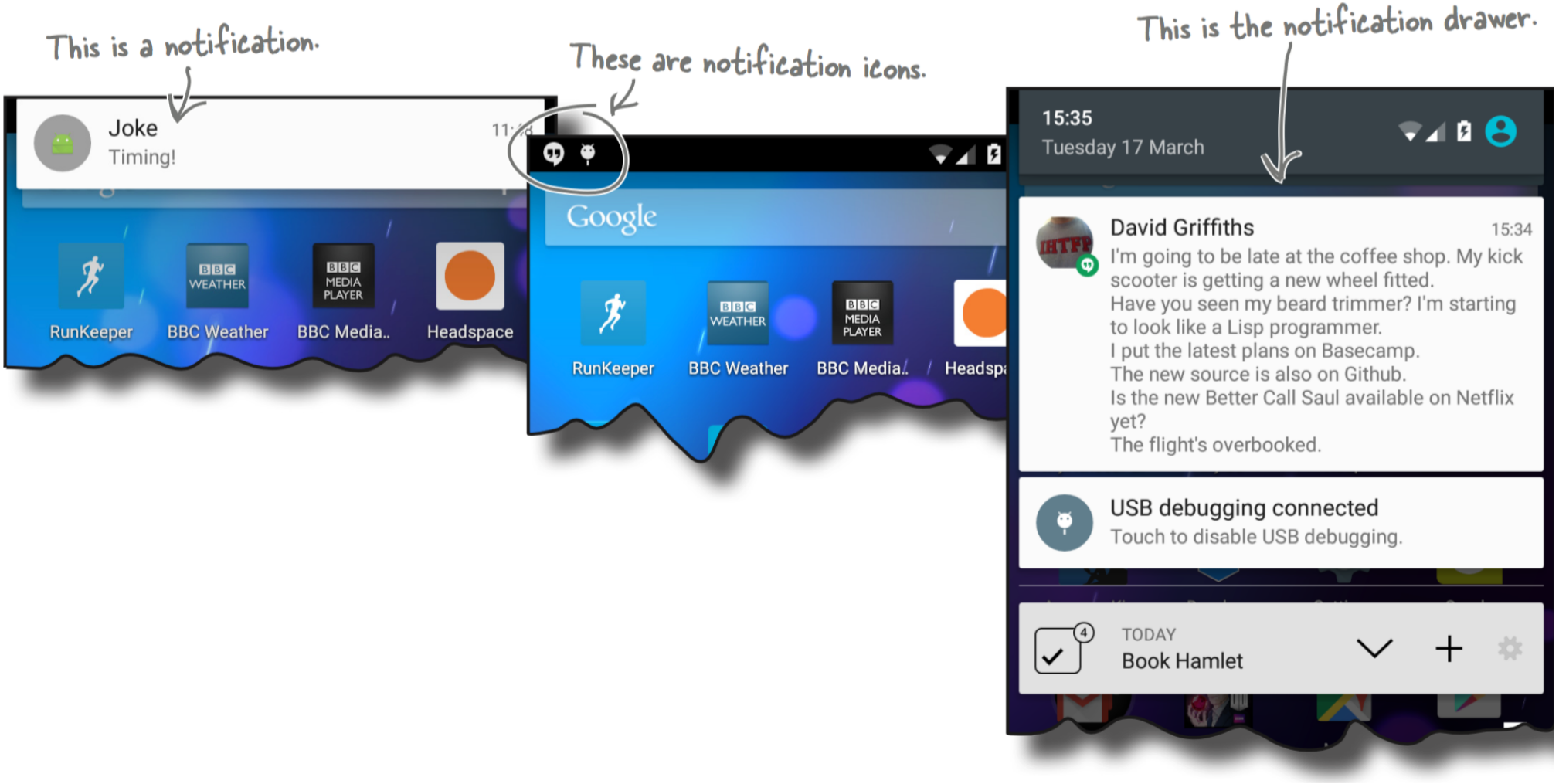


El código completo de DelayedMessageService

Prueba

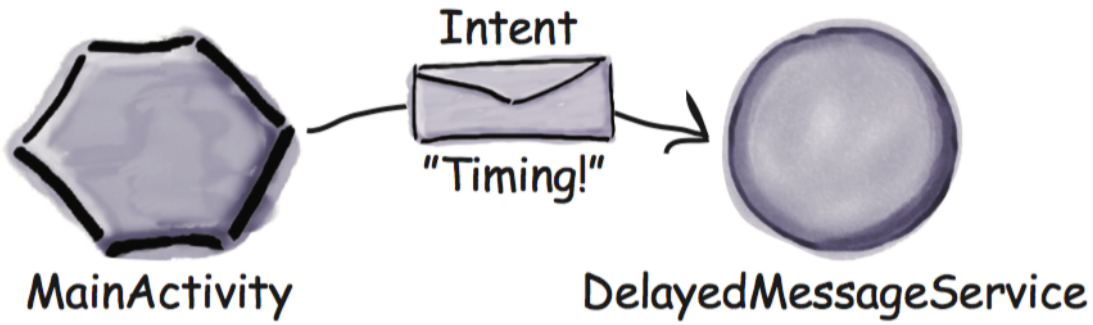

16:20
WHAT IS THE SECRET OF COMEDY?
Click on the button and wait.
10 seconds later, the toast appears.
Timing!

Mejorando las notificaciones



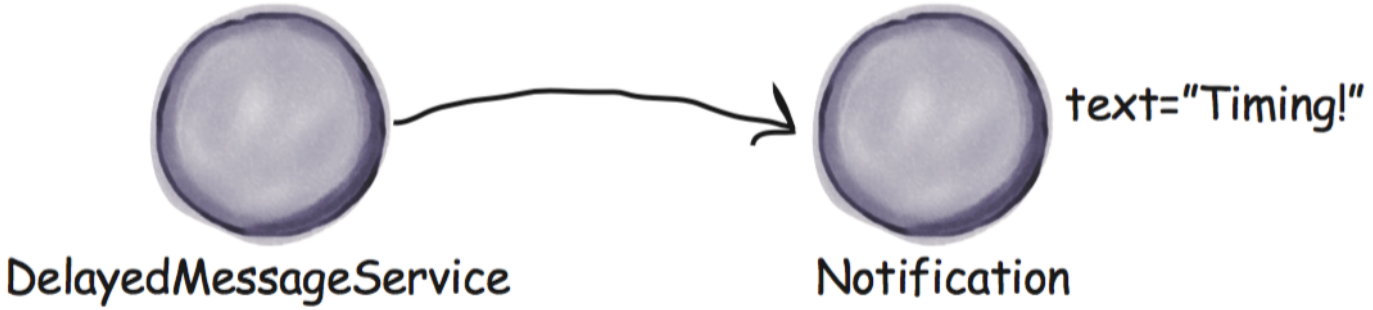
Como se utiliza el servicio de notificaciones

1 **MainActivity starts DelayedMessageService by passing it an intent.**



2 **DelayedMessageService creates a new Notification object.**

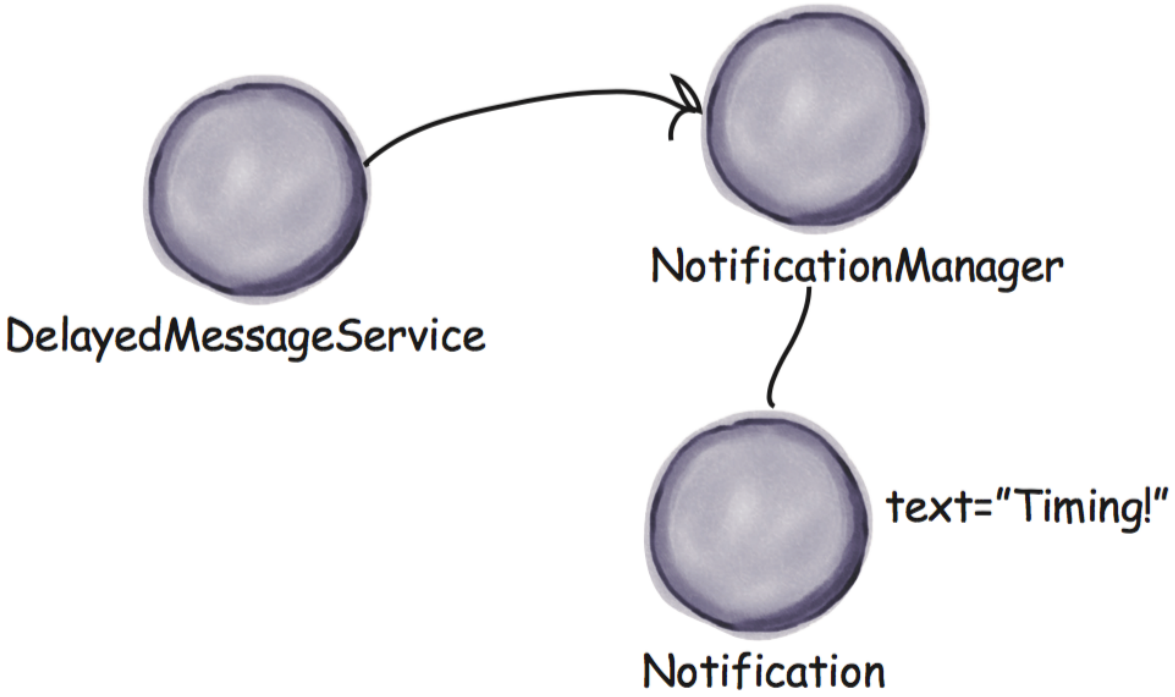
The `Notification` object contains details of how the notification should be configured, such as its text, title, and icon.



Como se utiliza el servicio de notificaciones

3 **DelayedMessageService creates a NotificationManager object to access Android's notification service.**

`DelayedMessageService` passes the `Notification` object to the `NotificationManager`, and the notification gets displayed.



Creando notificaciones con Notification.Builder

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.

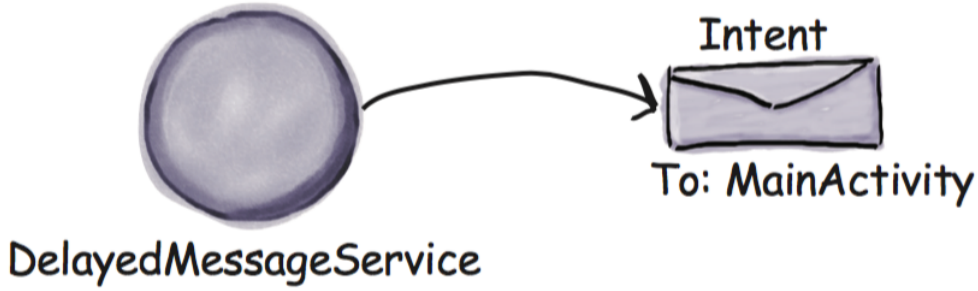
Notificaciones para iniciar actividades

1. Create an explicit intent

First, you create a simple explicit intent directed to the activity you want to start when the notification is clicked. In our case, we'll start `MainActivity`:

This is a normal intent that starts MainActivity.

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



Notificaciones para iniciar actividades

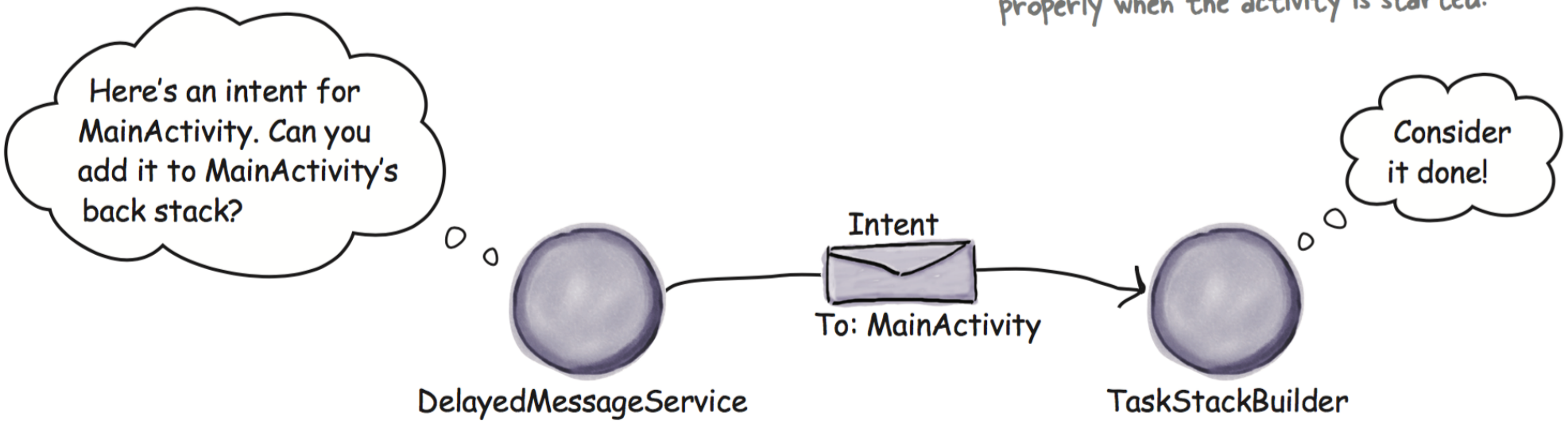
2. Pass the intent to the TaskStackBuilder

Next, we use a `TaskStackBuilder` to make sure that the back button will play nicely when the activity gets started. The `TaskStackBuilder` allows you to access the history of activities used by the back button. We need to get the back stack related to the activity, and then add the intent we just created to it:

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.



3. Get the pending intent from the TaskStackBuilder

Next, we get the pending intent from the `TaskStackBuilder` using its `getPendingIntent()` method. The `getPendingIntent()` method takes two `int` parameters, a request code that can be used to identify the intent, and a flag that specifies the pending intent's behavior.

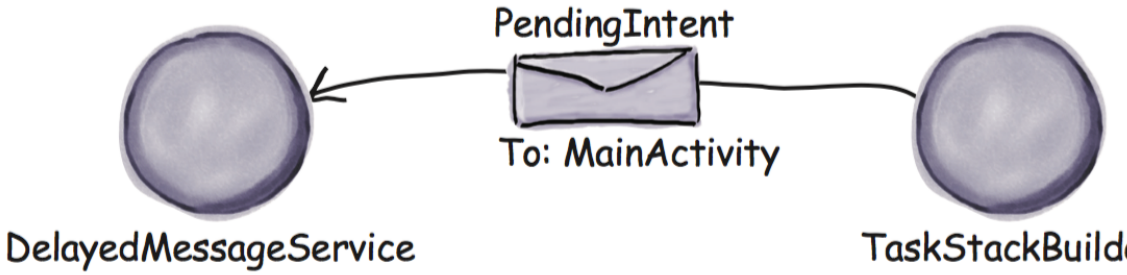
Here are the different flag options:

`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.

In our case, we'll use FLAG_UPDATE_CURRENT to modify any existing pending intent. Here's the code:

This creates the pending intent.

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



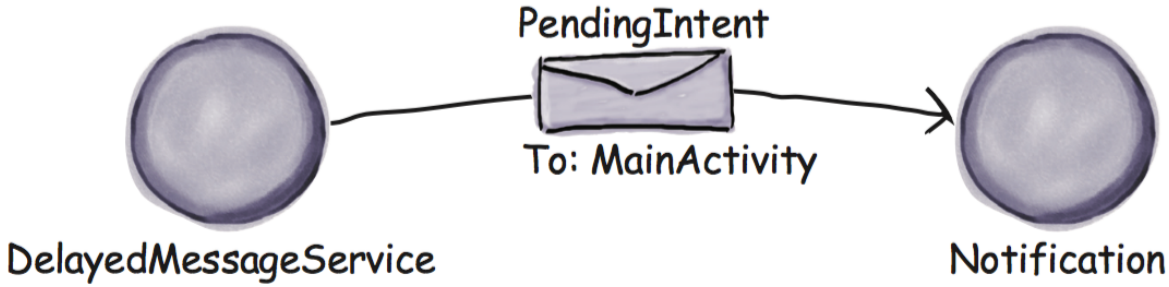
Notificaciones para iniciar actividades

4. Add the intent to the notification

Finally, you add the pending intent to the notification using the `setContentIntent()` method:

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

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



Enviando la notificación

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

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

This is how you access Android's notification service.

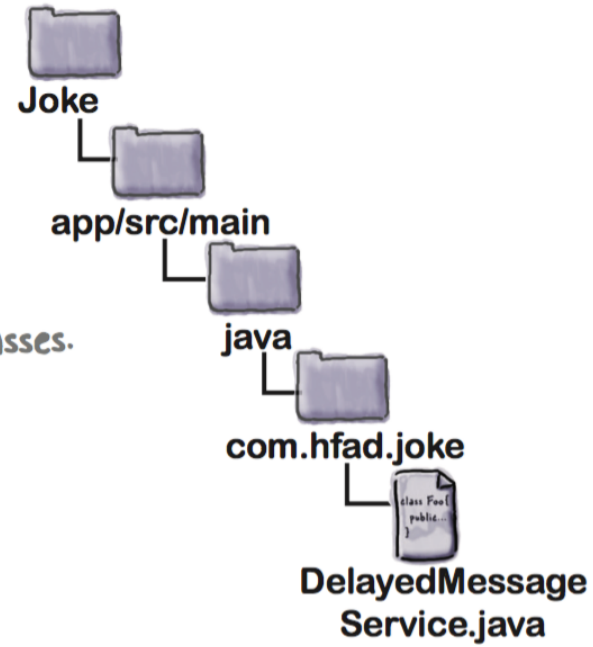
Use the notification service to display the notification we created.

El código completo de DelayedMessageService

```
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;
```

We're using these extra classes.

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



```
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 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.

El código completo de DelayedMessageService

El código completo de DelayedMessageService

```
@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.

El código completo de DelayedMessageService

```
        //Create a notification builder
        NotificationCompat.Builder builder =
                new NotificationCompat.Builder(this)
                        .setSmallIcon(android.R.drawable.sym_def_app_icon)
                        .setContentTitle(getString(R.string.question))
                        .setContentText(text)
                        .setPriority(NotificationCompat.PRIORITY_HIGH)
                        .setVibrate(new long[] {0, 1000})
                        .setAutoCancel(true);

        //Create an action
        Intent actionIntent = new Intent(this, MainActivity.class);
        PendingIntent actionPendingIntent = PendingIntent.getActivity(
                                            this,
                                            0,
                                            actionIntent,
                                            PendingIntent.FLAG_UPDATE_CURRENT);
        builder.setContentIntent(actionPendingIntent);

        //Issue the notification
        NotificationManager notificationManager =
                (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        notificationManager.notify(NOTIFICATION_ID, builder.build());
    }
}
```

Use a notification builder to specify the content and features of the notification.

Create an intent.

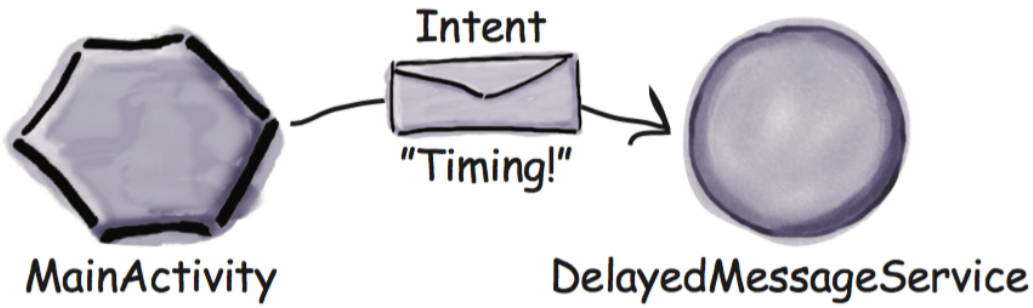
Use the intent to create a pending intent.

Add the pending intent to the notification.

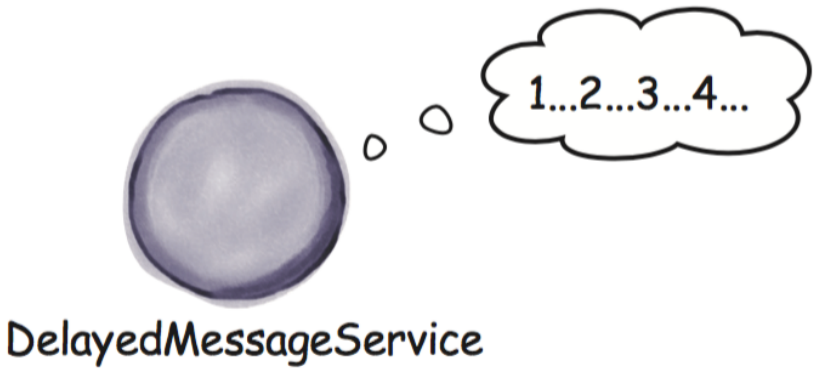
Display the notification using a notification manager.

Que pasa al ejecutar el código

1 **MainActivity starts DelayedMessageService by passing it an intent.**
The intent contains the message `MainActivity` wants `DelayedMessageService` to display.

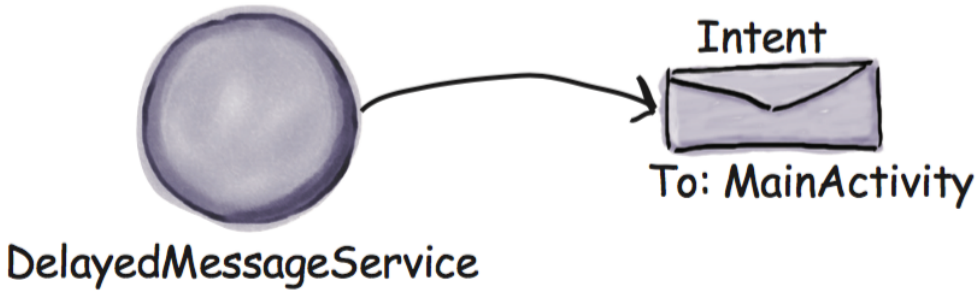


2 **DelayedMessageService waits for 10 seconds.**

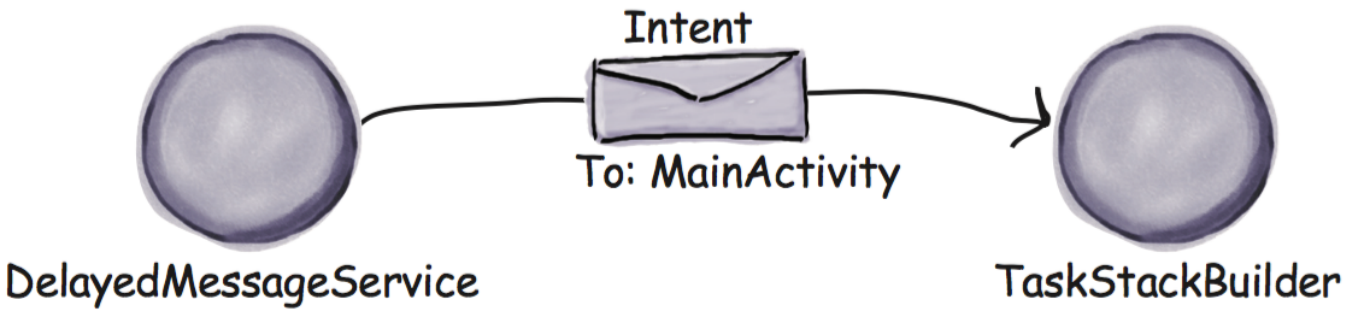


Que pasa al ejecutar el código

3 **DelayedMessageService creates an intent for MainActivity.**

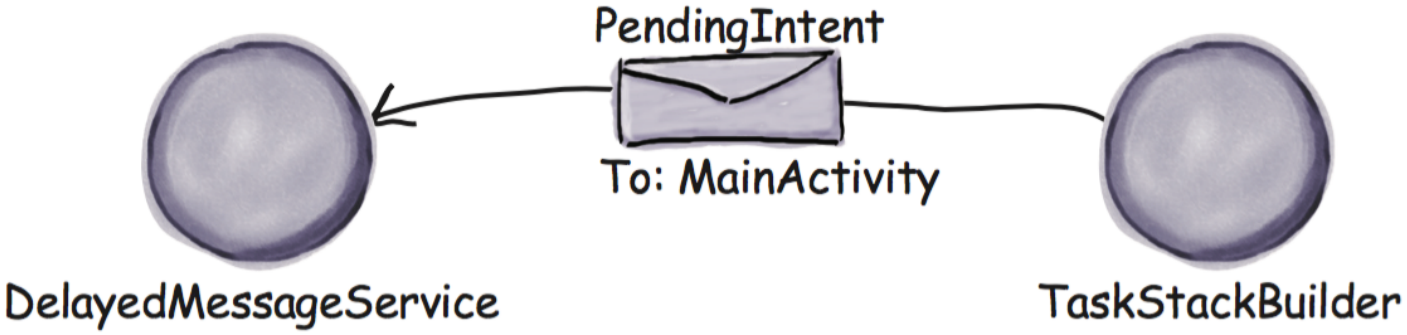


4 **DelayedMessageService creates a TaskStackBuilder and asks it to add the intent to MainActivity's back stack.**

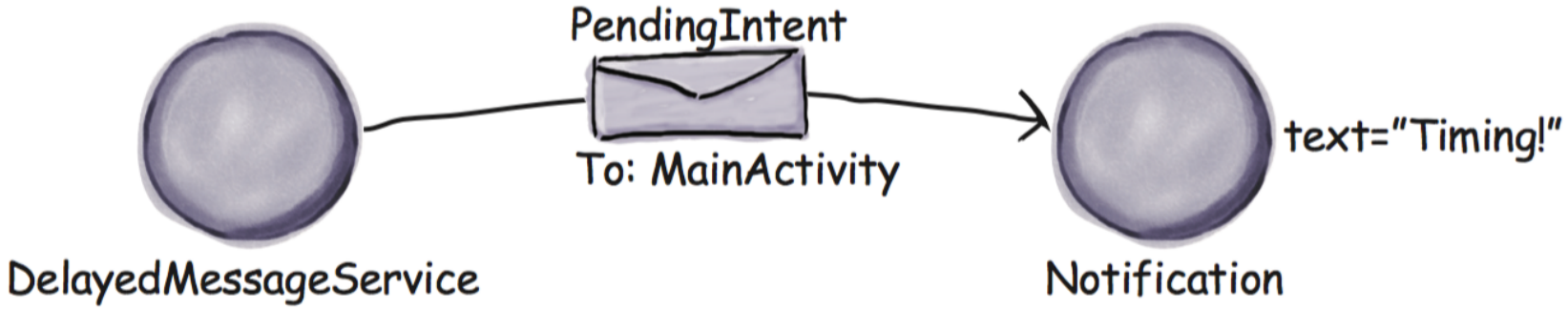


Que pasa al ejecutar el código

5 **The TaskStackBuilder use the intent to create a pending intent and passes it to DelayedMessageService.**

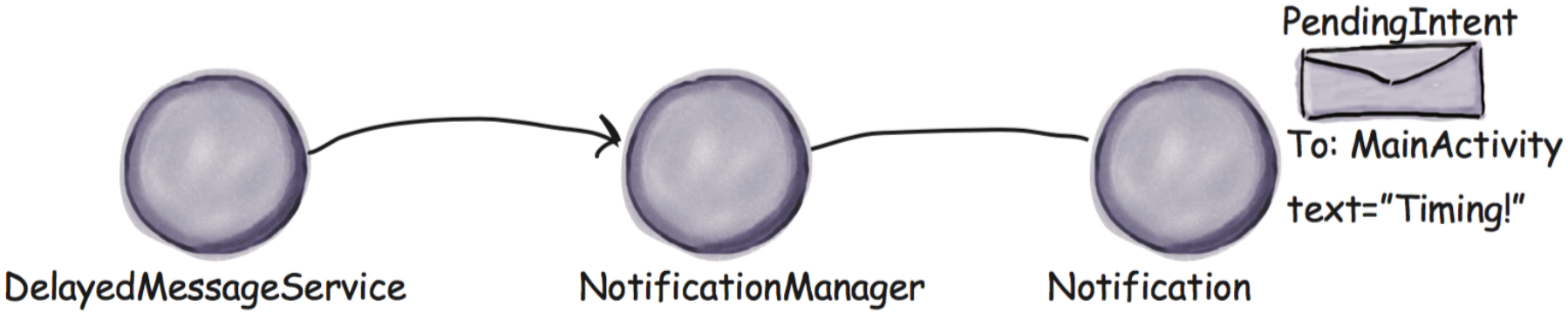


6 **DelayedMessageService creates a Notification object, sets details of how it should be configured, and passes it the pending intent.**

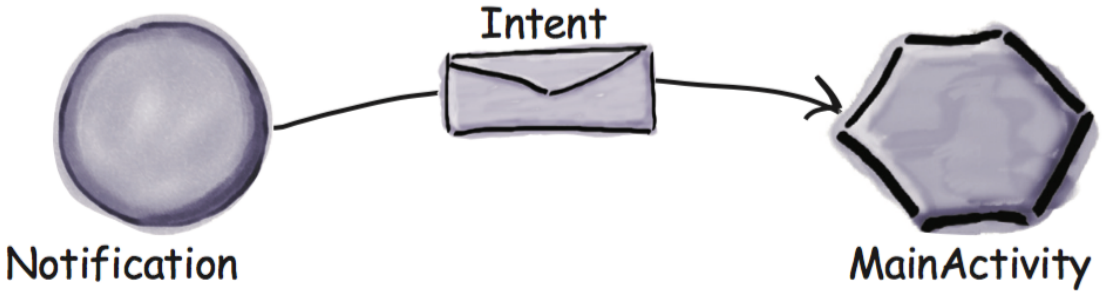


Que pasa al ejecutar el código

7 **DelayedMessageService creates a NotificationManager object to access Android's notification service and passes it the Notification.**
The notification service displays the notification.



8 **When the user clicks on the Notification, the Notification uses its pending intent to start MainActivity.**



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

Prueba

Service Magnets Solution

Below you'll see most of the code needed to create a started service called `WombleService` that plays a *.mp3* file in the background, and an activity that uses it. See if you can finish off the code.

```
public class WombleService extends [IntentService] {

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

    @Override
    protected void [onHandleIntent](Intent intent) {
        MediaPlayer mediaPlayer =
                MediaPlayer.create(getApplicationContext(), R.raw.wombling_song);
        mediaPlayer.start();
    }
}
```

This is the service. It extends the IntentService class.

The code needs to run in the onHandleIntent() method.

```
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, WombleService.class);
        startService(intent);
    }
}
```

This is the activity.

Create an explicit intent directed at WombleService.class.

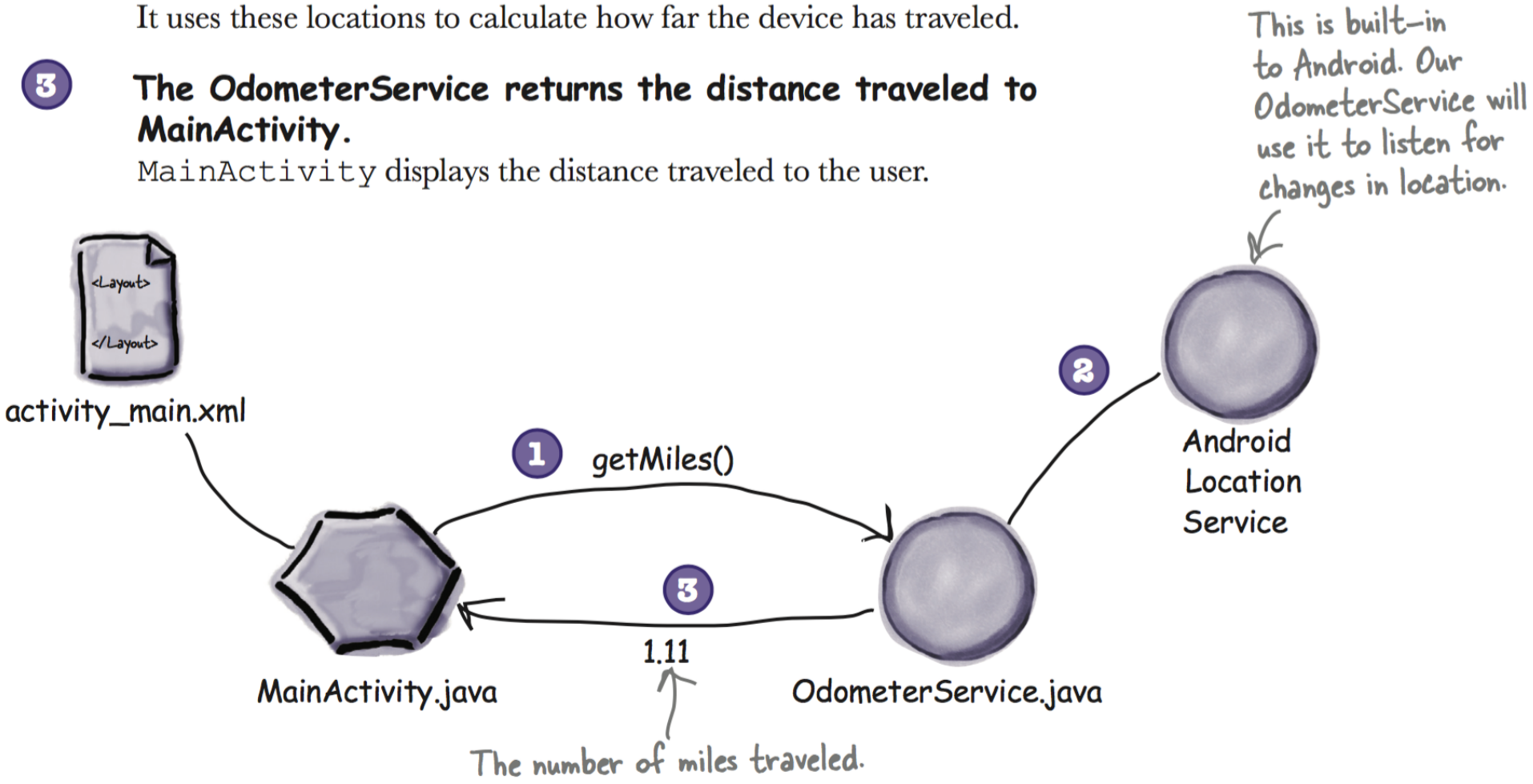
Start the service.

Los componentes atados son más interactivos

1. **MainActivity binds to OdometerService.**
   `MainActivity` uses the `OdometerService` `getMiles()` method to ask for the number of miles traveled.

2. **The OdometerService uses the Android location services to keep track of when the device moves.**
   It uses these locations to calculate how far the device has traveled.

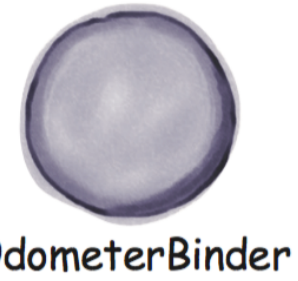
3. **The OdometerService returns the distance traveled to MainActivity.**
   `MainActivity` displays the distance traveled to the user.



Los pasos para crear el servicio de odómetro

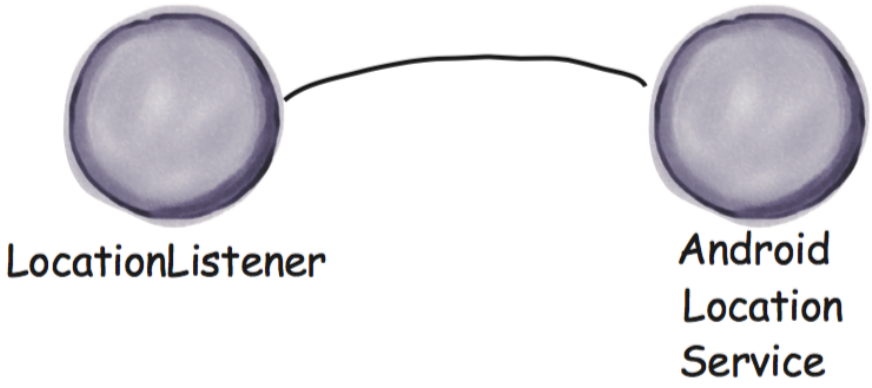
1 **Define an OdometerBinder.**

A `Binder` object allows activities to bind to services. We'll define a subclass of `Binder` called `OdometerBinder` that will enable our activity to connect to the `OdometerService`.



2 **Create a LocationListener and register it with Android's location service.**

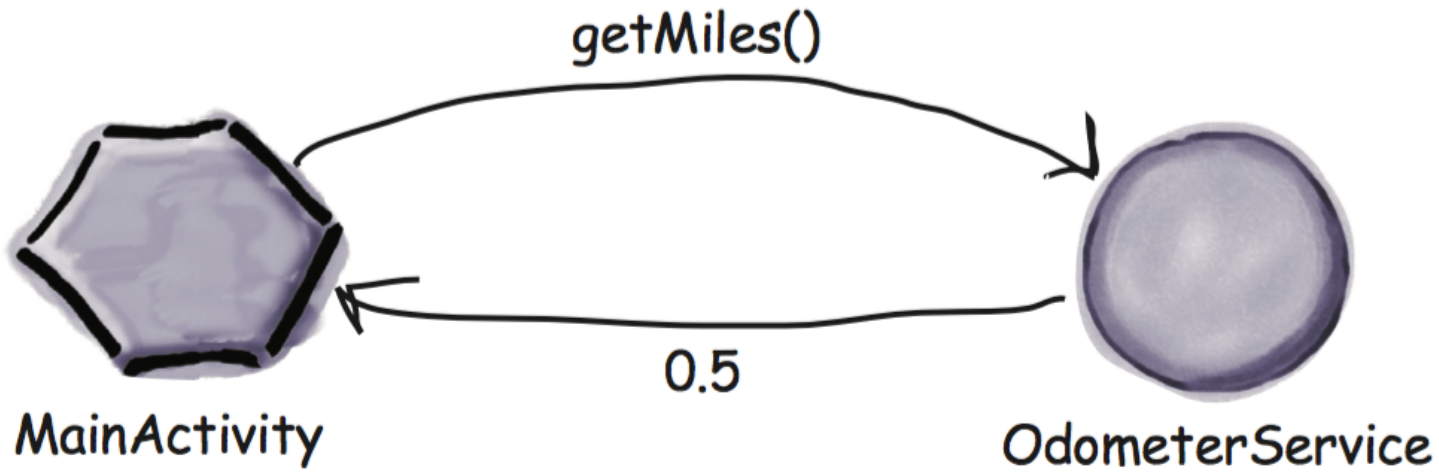
This will allow the `OdometerService` to listen for changes in the device location and work out the distance traveled in meters.



Los pasos para crear el servicio de odómetro

3 **Create a public getMiles() method.**

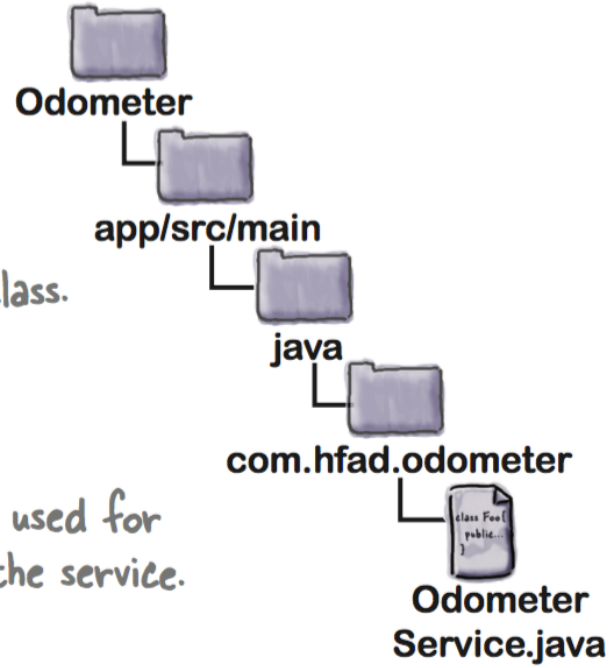
The activity will be able to use this to get the number of miles traveled.

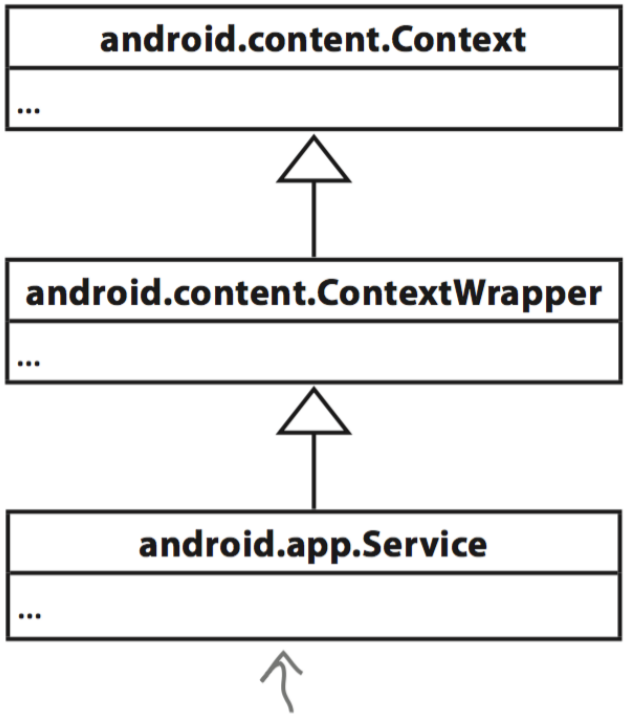


```
package com.hfad.odometer;

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.





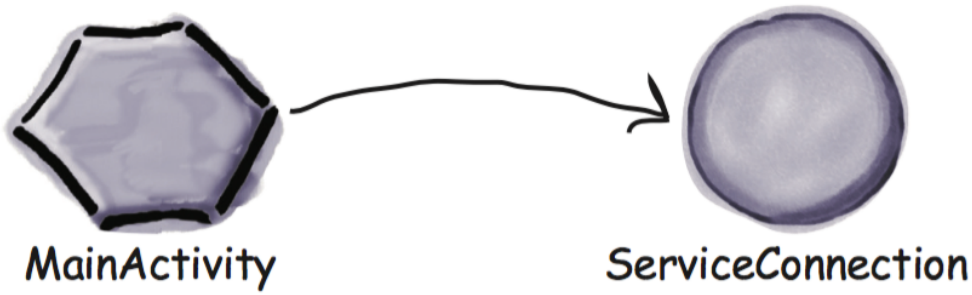
Here's the class hierarchy for the Service class.

Creando el nuevo proyecto de odómetro

Como funciona el atado

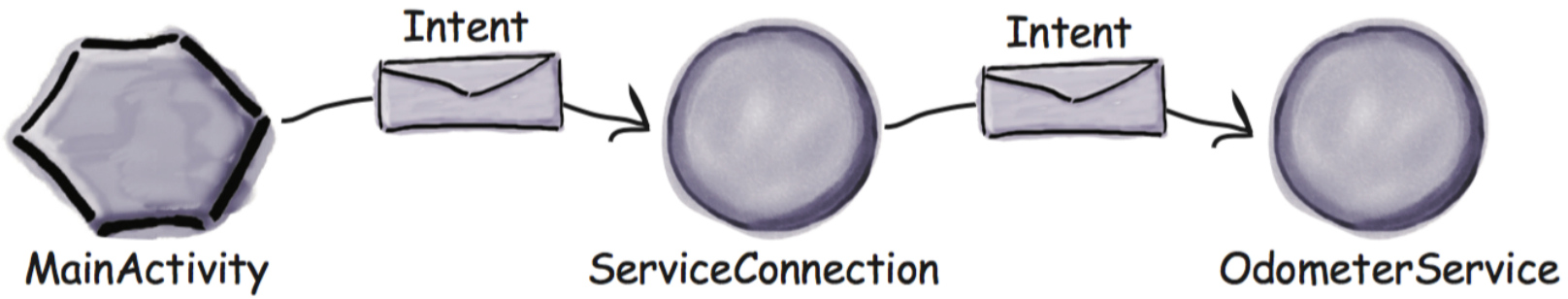
1 **The activity creates a ServiceConnection object.**

A `ServiceConnection` is used to form a connection with the service.



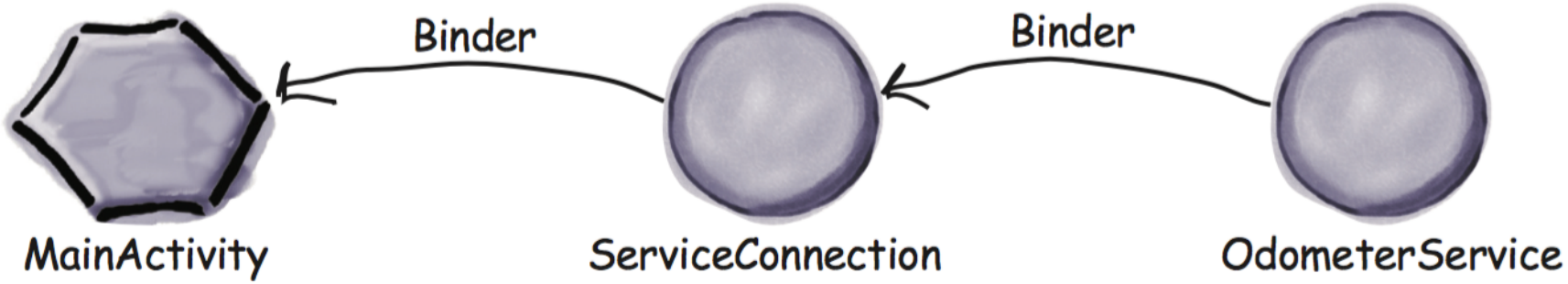
2 **The activity passes an Intent down the connection to the service.**

The intent contains any additional information the activity needs to pass to the service.

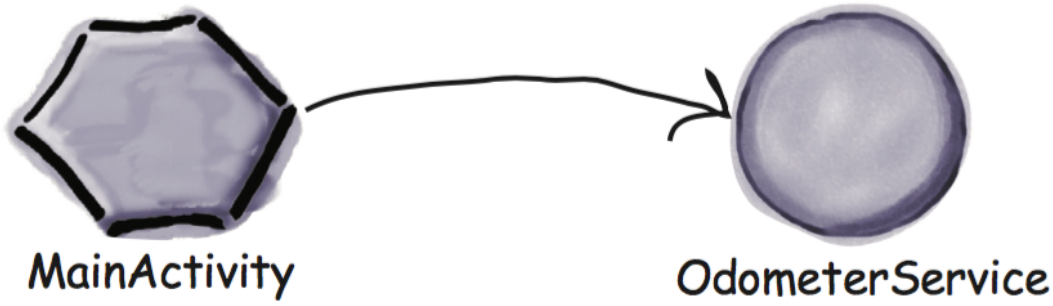


Como funciona el atado

3 **The bound service creates a Binder object.**
The Binder contains a reference to the bound service. The service sends the Binder back along the connection.



4 **When the activity receives the Binder, it takes out the Service object and starts to use the service directly.**



Definiendo el atado

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

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

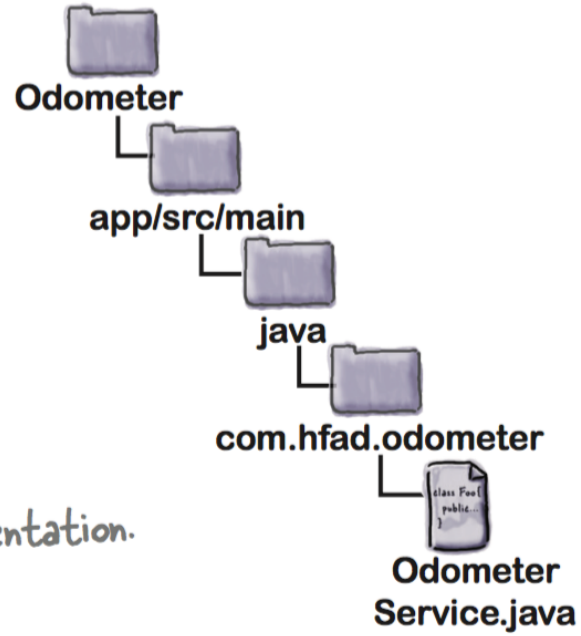
The activity will use this method to get a reference to the OdometerService.

```
...
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;
    }
}
```

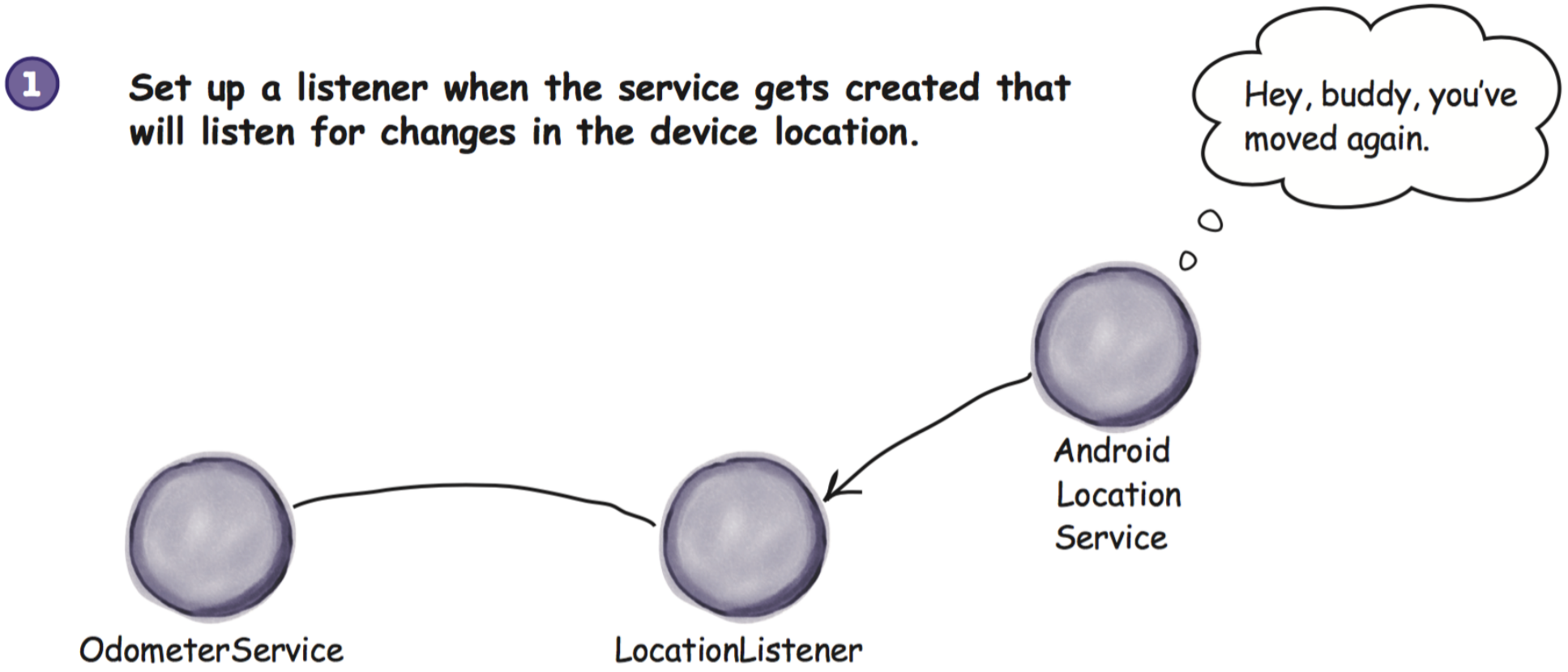
We're using these classes.

The Binder implementation.

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

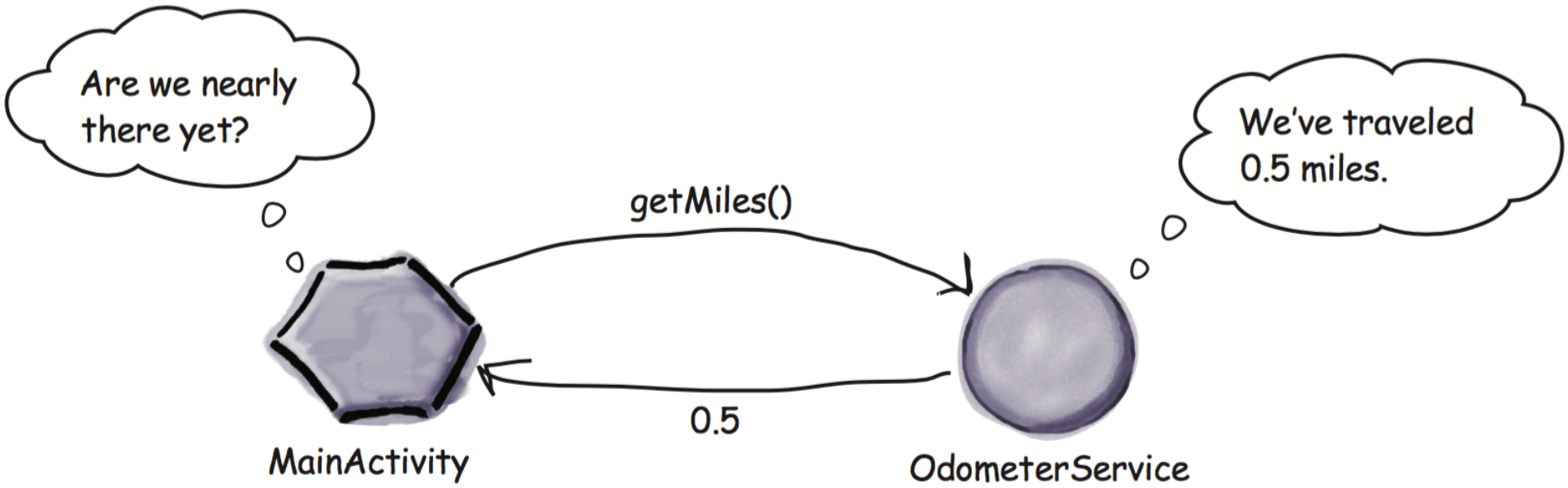


Obteniendo el servicio para hacer algo



Obteniendo el servicio para hacer algo

2 **Return the number of miles traveled to the activity whenever the activity asks for it.**



Métodos de la clase Service

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.

Localización

```
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 is the new LocationListener.

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.

Agregando el LocationListener

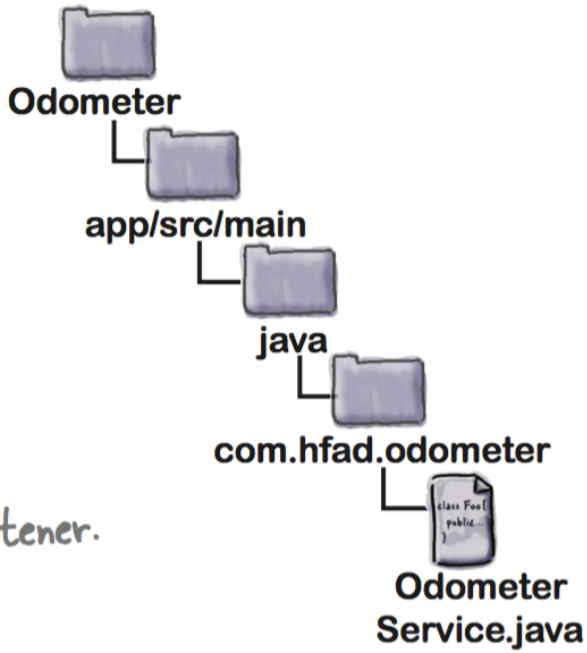
```
...
public class OdometerService extends Service {

    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;
            }
```

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

Create the listener.

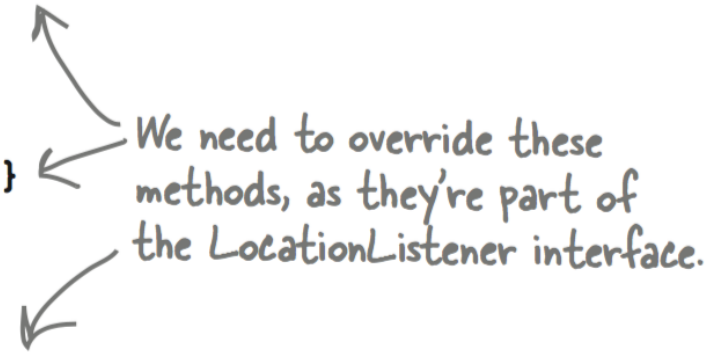
If it's our first location, set lastLocation to the current Location.

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



Agregando el LocationListener

```
        @Override
        public void onProviderDisabled(String arg0) {}

        @Override
        public void onProviderEnabled(String arg0) {}

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



Registrando el LocationListener

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

This is the GPS provider.

The time in milliseconds.

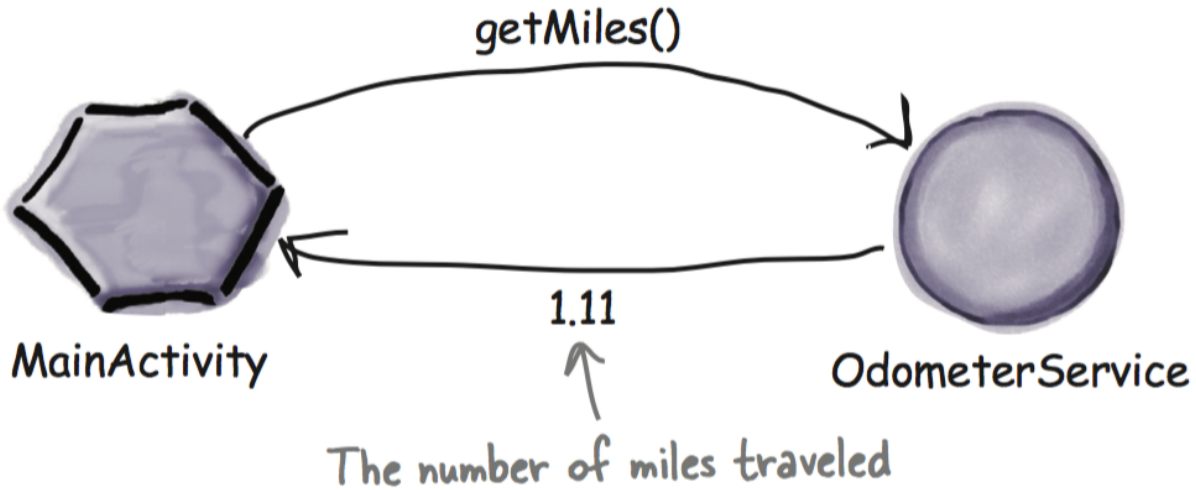
The distance in meters.

This is the LocationListener we created.

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

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

Distancia viajada



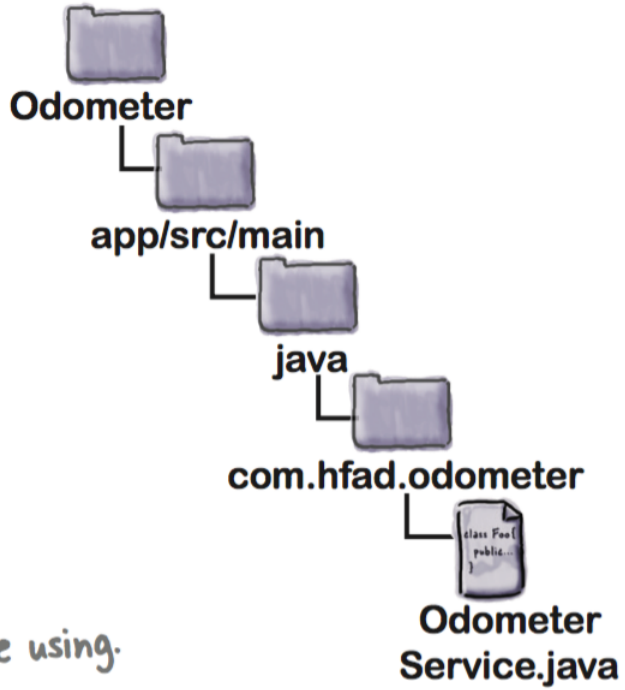

```
public double getMiles() {
    return this.distanceInMeters / 1609.344;
}
```

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.

El código completo de OdometerService

```
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;
```

These are all the classes we're using.



El código completo de OdometerService

```
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 the private variables we're using.

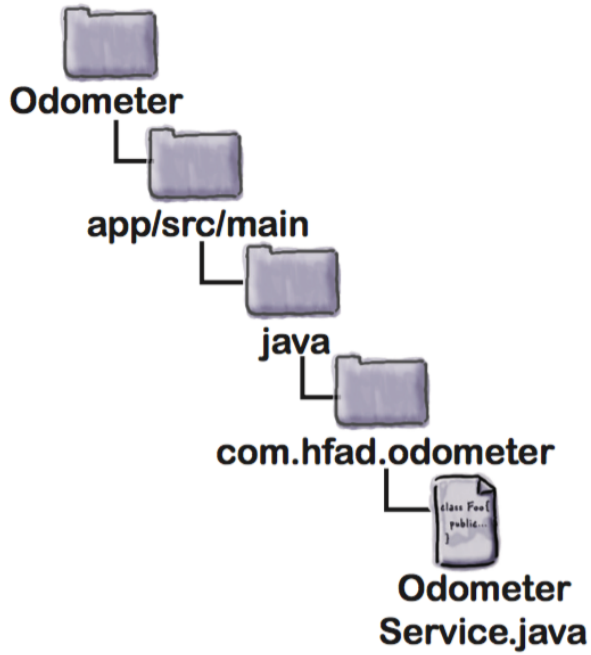
When you create a bound service, you have to define a Binder object. It enables the activity to bind to the service.

This gets called when the activity binds to the service.

```
@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) {}
    };
```

Set up the location listener when the service is created.

This is our implementaton of the location listener.



El código completo de OdometerService

El código completo de OdometerService

```
        LocationManager locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
        locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, listener);
    }

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

Convert the distance traveled to miles and return it.

Register the location listener with the location service.

Actualizando AndroidManifest.xml

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>
```

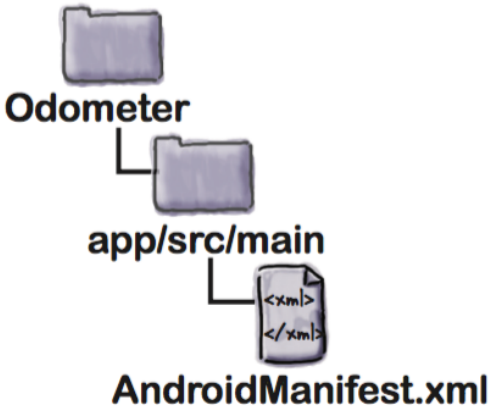
Actualizando AndroidManifest.xml

```
<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.



Service Magnets Solution

See if you can complete the code below to create a bound service called `NumberService` that returns a random number when its `getNumber()` method is called.

```
...
public class NumberService extends Service {

    private final IBinder binder = new NumberBinder();
    private final Random random = new Random();

    public class NumberBinder extends Binder {
        NumberService getNumberService() {
            return NumberService.this;
        }
    }

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

    public int getNumber() {
        return random.nextInt(100);
    }
}
```

Define a NumberBinder class that extends Binder.

The activity needs to get a reference to the NumberService from the Binder, so it needs to return a NumberService object.

Override the onBind() method so the activity can bind to the service.

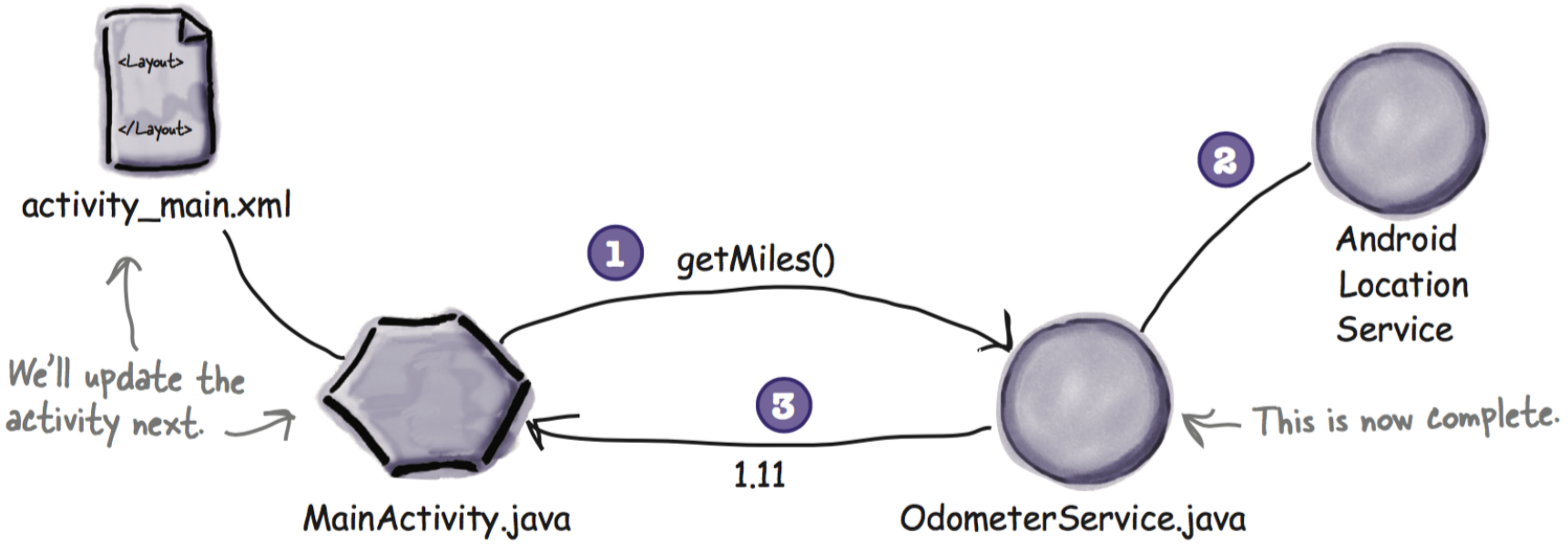
The onBind() method should return the Binder.

En donde estamos ?

1. **MainActivity binds to OdometerService.**
   `MainActivity` uses the `OdometerService` `getMiles()` method to ask for the number of miles traveled.

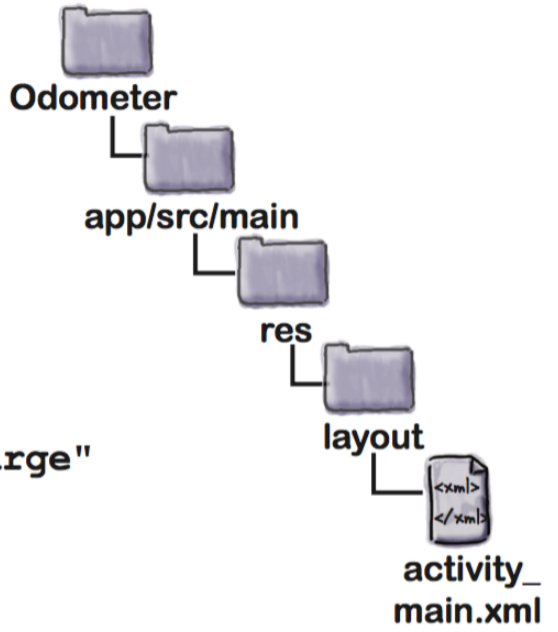
2. **The OdometerService uses the Android location services to keep track of when the device moves.**
   It uses these locations to calculate how far the device has traveled.

3. **The OdometerService returns the distance traveled to MainActivity.**
   `MainActivity` displays the distance traveled to the user.

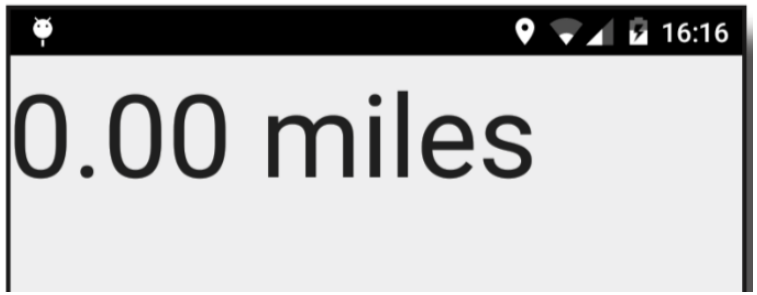


Actualizando el layout de la actividad principal

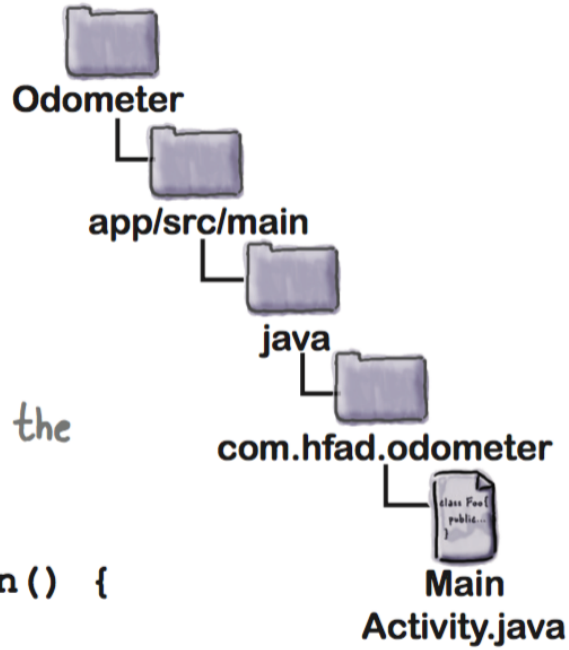
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <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.



Creando un servicio de conexión




```
...
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;
        }
    };
}
```

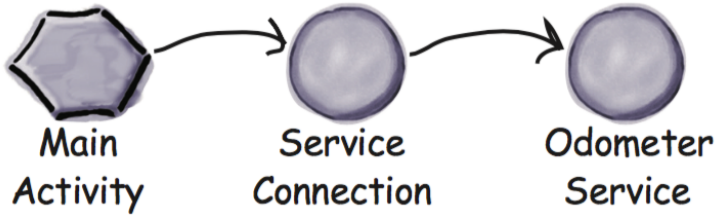
We'll use this for the OdometerService.

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

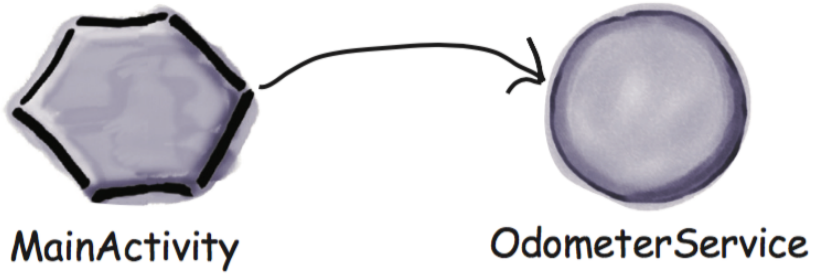
Cast the Binder to an OdometerBinder, then use to get a reference to the OdometerService.

When the service is connected, set bound to true.

When the service is disconnected, set bound to false.



Atando el servicio al iniciar la actividad

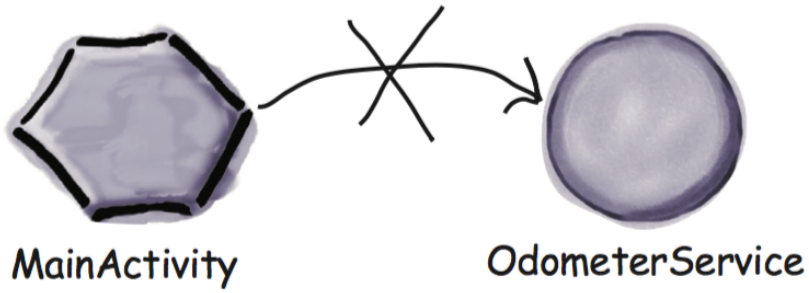



```
@Override
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, OdometerService.class);
    bindService(intent, connection, Context.BIND_AUTO_CREATE);
}
```

This is an intent directed to the OdometerService.

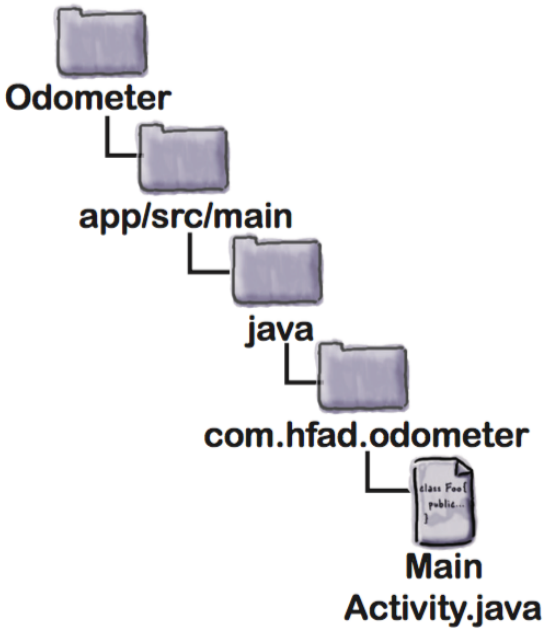
This uses the intent and service connection to bind the activity to the service.

Desatando el servicio al detener la actividad




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

This uses the service connection to unbind from the service.



```
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.

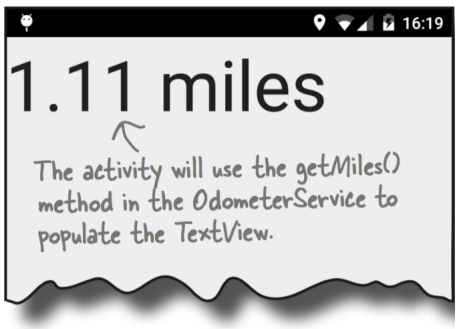
Create a new Handler.

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.

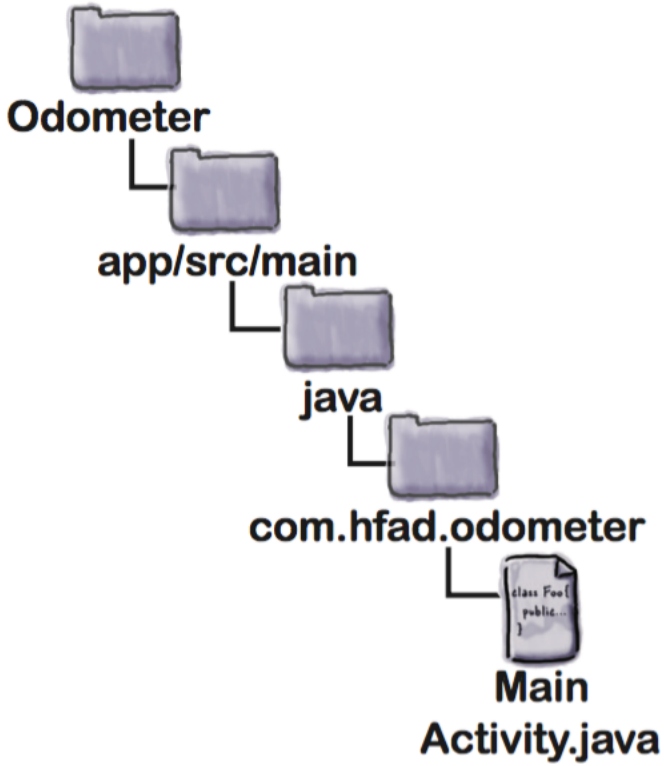
Post the code in the Runnable to be run again after a delay of 1,000 milliseconds, or 1 second. As this line of code is included in the Runnable run() method, it will run every second (with a slight lag).



Mostrando la distancia viajada

El código completo de MainActivity

```
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.

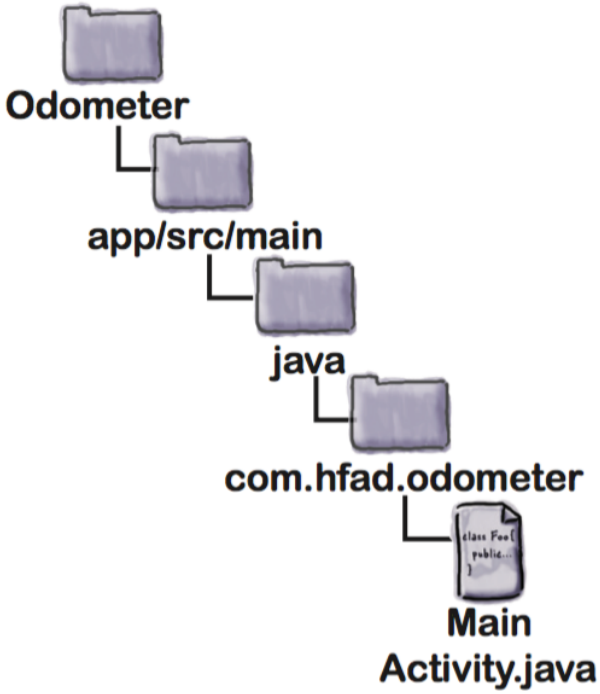
El código completo de MainActivity

El código completo de MainActivity

```
@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;
    }
}
```

Bind the service when the activity starts.

Unbind the service when the activity stops.



El código completo de MainActivity

This method updates the mileage that's displayed.

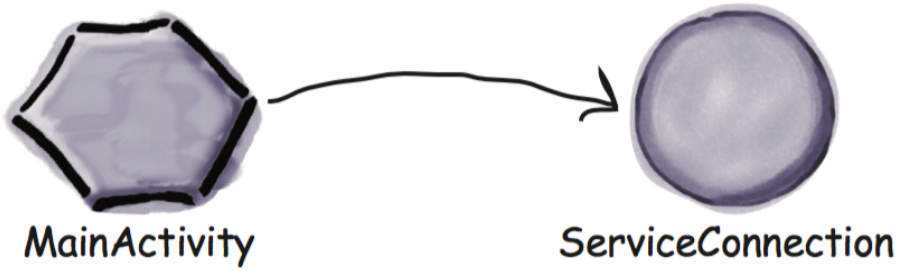
```
private void watchMileage() {
    final TextView distanceView = (TextView)findViewById(R.id.distance);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            double distance = 0.0;
            if (odometer != null) {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

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

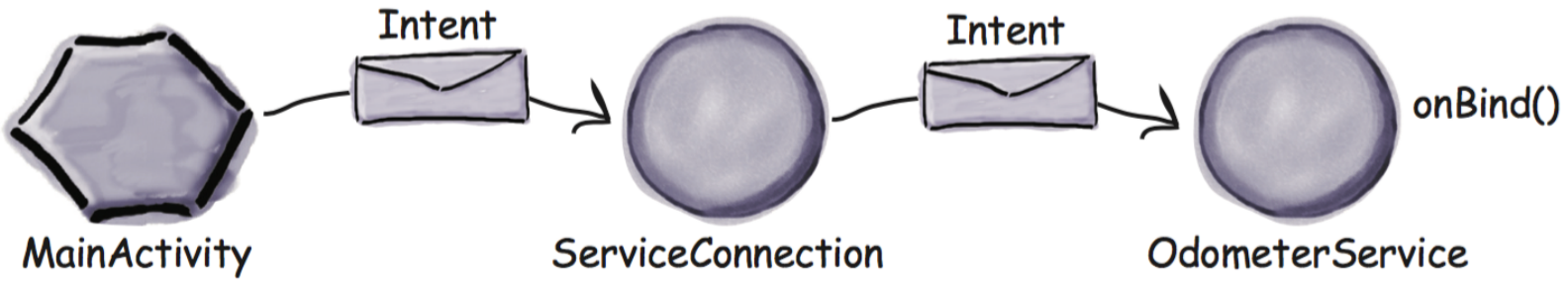
Update the distance every second.

Que pasa al ejecutar el código

1 **When the MainActivity starts, the onStart() method creates a ServiceConnection.**
It asks to bind to the `OdometerService`.

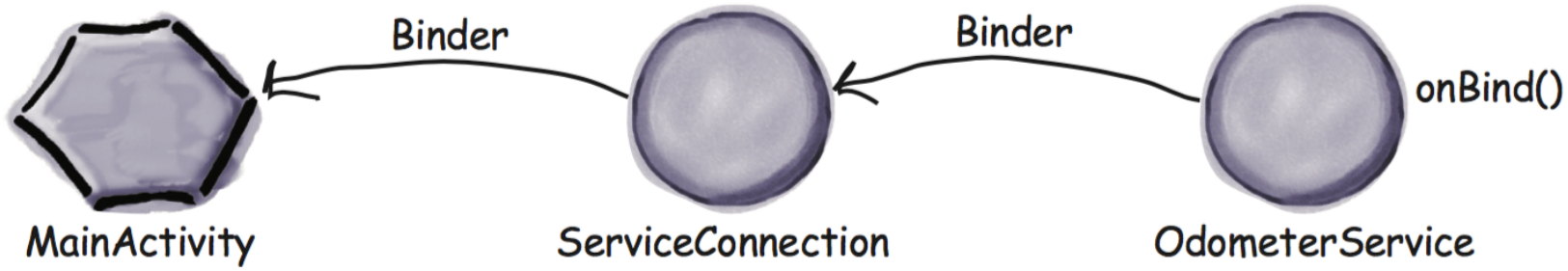


2 **The OdometerService starts and its onBind() method is called with a copy of the intent from the MainActivity.**

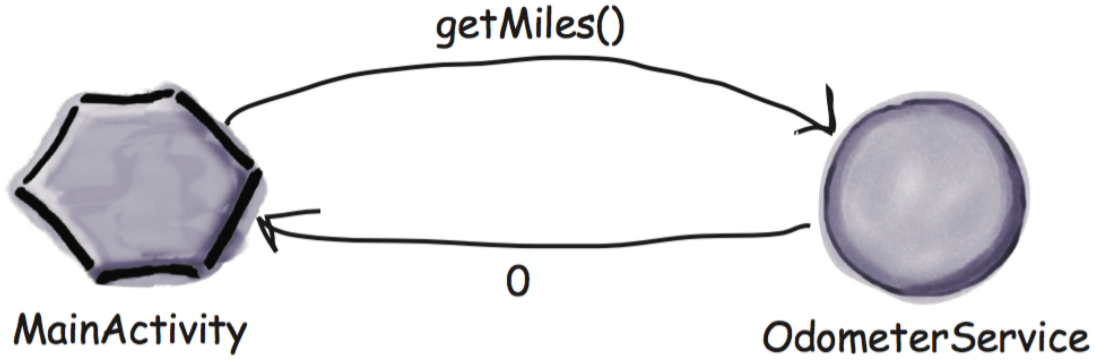


Que pasa al ejecutar el código

3 **The onBind() method returns a Binder object.**

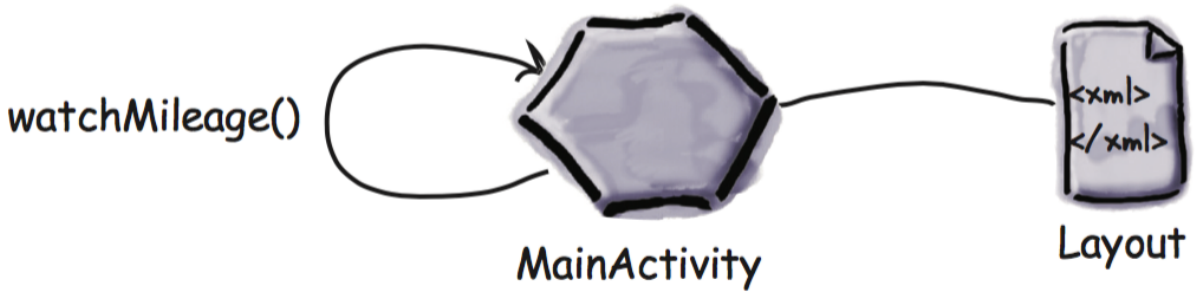


4 **MainActivity gets a reference to OdometerService from the Binder and starts to use the service directly.**

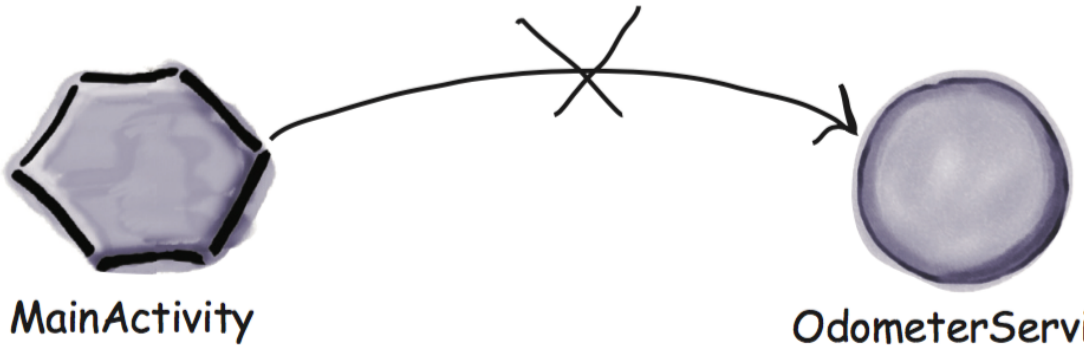


Que pasa al ejecutar el código

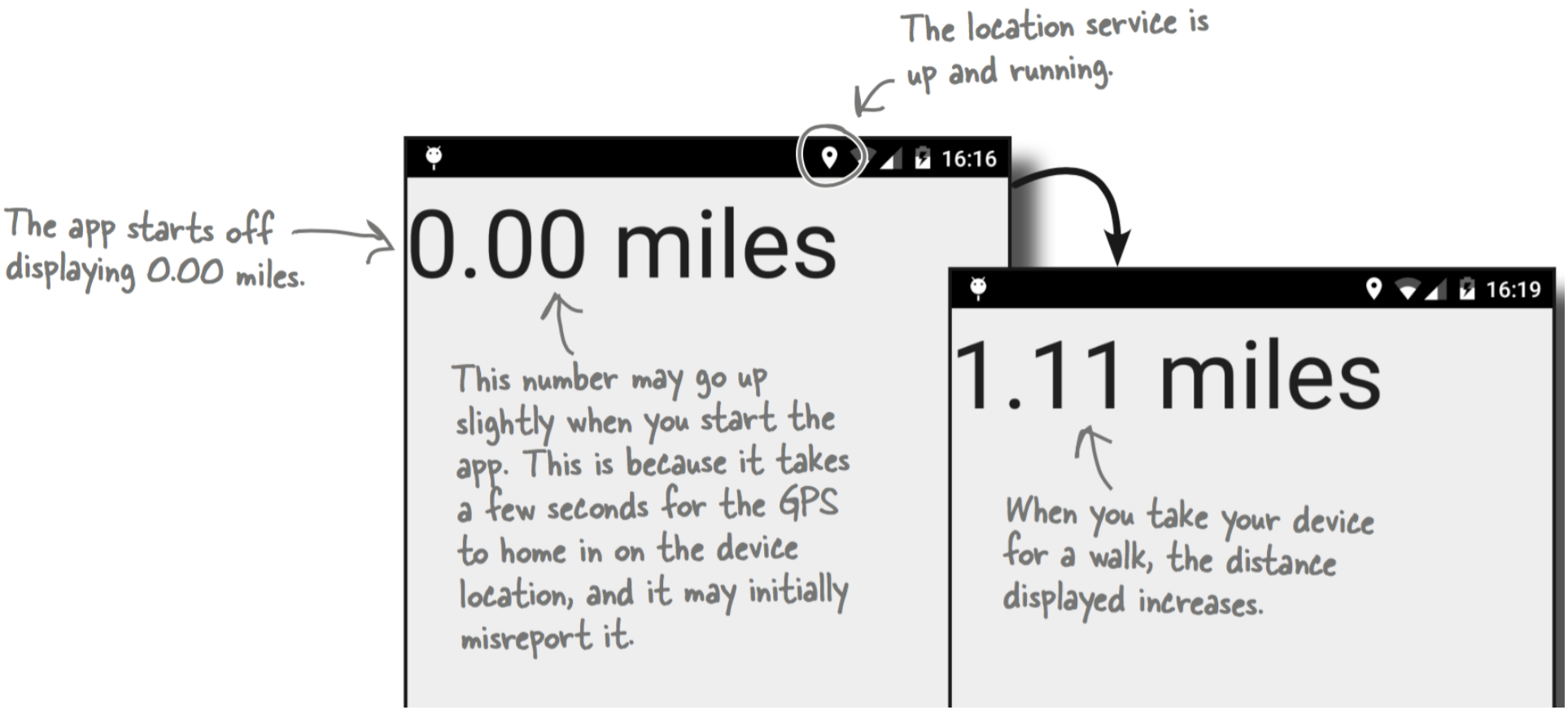
5 **While MainActivity is running, the watchMileage() method calls the OdometerService getMiles() method every second and updates the screen.**



6 **When MainActivity stops, it disconnects from the OdometerService by calling unbindService().**



Prueba


The location service is up and running.
16:16
The app starts off displaying 0.00 miles.
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 `IntentService` class 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.
- 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.