Understanding Android Pending Intents

PendingIntent is basically an object that wraps another Intent object. Then it can be passed to a foreign application where you're granting that app the right to perform the operation, i.e., execute the intent as if it were executed from your own app's process (same permission and identity). For security reasons you should always pass explicit intents to a PendingIntent rather than being implicit.

It can be used for various purposes like declaring an Intent which is executed at some point of time in future via the `AlarmManager` class or one which is executed when the user performs some action with your app Notification.

A PendingIntent itself is a token referencing the original data (Intent action, data, categories, etc.) maintained by the system. Hence if the owning app's (creator) process is killed, the PendingIntent will still remain usable from other app proesses that had received it. The owning app can later re-retrieve the PendingIntent token if that's still valid by specifying the same set of data and then even `cancel()` it.

Each explicit intent is supposed to be handled by a specific app component like Activity, BroadcastReceiver or a Service. Hence PendingIntent has different methods to handle different types of Intents that it wraps:

- `PendingIntent.getActivity()` – Retrieve a PendingIntent to start an Activity.
- `PendingIntent.getBroadcast()` – Retrieve a PendingIntent to perform a broadcast.
- `PendingIntent.getService()` – Retrieve a PendingIntent to start a Service.

Let's get into some code now. We'll create an Intent and wrap it into a PendingIntent:

```
1
2     // Explicit intent to wrap
      Intent intent = new Intent(this, LoginActivity.class);
3
4     // Create pending intent and wrap our intent
5     PendingIntent pendingIntent = PendingIntent.getActivity
6     (this, 1, intent, PendingIntent.FLAG_CANCEL_CURRENT);
      try {
7         // Perform the operation associated with our pendingIntent
8         pendingIntent.send();
9     } catch (PendingIntent.CanceledException e) {
10        e.printStackTrace();
      }
11
```

So we created an intent, wrapped it inside a PendingIntent and executed the operation associated with the pending intent (using `send()` which is invoking `LoginActivity`. Let's once go through the arguments passed to `getActivity()`:

- `this` (context) – This is the context in which the `PendingIntent` should start the activity.
- `1` (requestCode) – This is a private request code for the sender. Use it later with the same method again to get back the same pending intent later. Then you can do various things like cancelling the pending intent with `cancel()`, etc.
- `intent` (intent) – Explicit intent of the activity to be launched.
- `PendingIntent.FLAG_CANCEL_CURRENT` (flags) – One of the flags that can be passed that. This one states that if the described PendingIntent already exists, then the current one will be cancelled first and then the new one will be generated. I've seen some code samples where simply `0` is passed which isn't the value of any flags which probably means no flag is passed and the behaviour isn't documented but I guess it's similar to `FLAG_CANCEL_CURRENT`.

  Although this code sample wasn't any useful as the same could have been done with just an Intent and `startActivity()`, let's see a better usage of pending intents.

```
1
2     int seconds = 3;
3     // Create an intent that will be wrapped in PendingIntent
4     Intent intent = new Intent(this, MyReceiver.class);
5
6     // Create the pending intent and wrap our intent
7     PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 1, intent, 0);
8
9     // Get the alarm manager service and schedule it to go off after 3s
10    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
11    alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis() + (seconds *
12
13    Toast.makeText(this, "Alarm set in " + seconds + " seconds", Toast.LENGTH_LONG).sh
14
```

We kick off with in a similar fashion, i.e., create and Intent and wrap it inside a PendingIntent. Next we get the alarm manager to which we pass the `pendingIntent` and specify it to go off in 3 seconds. `AlarmManager.RTC_WAKEUP` passed to `AlarmManager.set()` is explained well here.

In your BroadcastReceiver's `onReceiver()` method you can add a simple piece of code like this:

```
1
2     @Override
3     public void onReceive(Context context, Intent intent) {
4         // Vibrate for 2 seconds
5         Vibrator vibrator = (Vibrator) context.getSystemService(Context.VIBRATOR_SERVIC
6         vibrator.vibrate(2000);
7     }
```

As soon as the broadcast event is sent, the device will vibrate for 2 seconds.