Google Android

GV: ThS. Phan Nguyệt Minh

minhpn@uit.edu.vn

http://courses.uit.edu.vn

Broadcast Receiver

Tổng quan

- BroadcastReceiver (có thể gọi là Receiver) là một trong bốn loại thành phần trong ứng dụng Android. Chức năng dùng để nhận các sự kiện mà các ứng dụng hoặc hệ thống phát đi.
- Có 2 cách phát-nhận đó là:
  - Không có thứ tự: receiver nào đủ điều kiện thì nhận hết, không phân biệt và cũng tách rời nhau.
  - Có thứ tự: receiver nào đăng ký ưu tiên hơn thì nhận trước, và có thể truyền thêm thông tin xử lý cho các receiver sau.

Lifecycle

- Thực ra lifecycle của BroadcastReceiver chỉ có duy nhất một phương thức onReceive().
  - Khi có sự kiện mà BroadcastReceiver đã đăng ký nhận được phát đi, thì phương thức onReceive() của BroadcastReceiver đó sẽ được gọi.
  - Sau khi thực thi xong phương thức này, lifecycle của Receiver kết thúc.

Lưu ý khi sử dụng

- Ngay khi onReceive() kết thúc, hệ thống coi như receiver đã không còn hoạt động và có thể kill process chứa receiver này bất cứ lúc nào.

→Tránh xử lý các code quá lâu trong onReceive().

→ Không có xử lý bất đồng bộ, chờ callback… trong Receiver (cụ thể như hiển thị Dialog, kết nối service…)

Một số broadcast thông dụng

- Báo hệ thống khởi động xong
- Báo pin có sự thay đổi
- Báo có package mới cài vào hoặc xóa đi
- Báo tắt máy
- Báo cắm sạc, rút sạc…

Một số broadcast khác

- Thông báo tin nhắn tới
- Thông báo cắm, rút thẻ nhớ
- Thông báo có cuộc gọi đi
- Có thể định nghĩa broadcast cho riêng mình (mục tiêu chính của việc này giúp liên lạc giữa các ứng dụng hoặc thông báo một sự kiện liên quan đến các ứng dụng khác)

onReceive()

- Phương thức này được gọi khi có sự kiện tương ứng được phát đi. Ở trong phương thức này, ta thấy truyền vào context và intent.
  - Vì Receiver không kế thừa từ Context nên cần truyền context mà receiver này đang chạy vào. Thứ nhất, để có thể xử lý các phương thức yêu cầu truyền thêm Context, thứ 2, để sử dụng các phương thức của lớp Context.

onReceive()

- Intent được truyền vào sẽ có đầy đủ thông tin như sự kiện nào mà receiver này đăng ký đã xảy ra dẫn đến onReceive() được gọi. Có gửi kèm thông tin gì hoặc dữ liệu gì hay không.

  Intent.getAction()

  Intent.get…Extra(String dataName)

Ví dụ BootReceiver

- Có thể đăng ký nhận sự kiện hệ thống vừa khởi động xong để có thể làm việc gì đó ngay, hoặc vận hành song song với hệ thống…
- Ta sẽ đăng ký nhận sự kiện BOOT_COMPLETED, sau đó sẽ gọi một dialog lên hiển thị lời chào.
- Khi hệ thống khởi động xong sẽ xuất một dialog chào user

Ví dụ BootReceiver

- Trong manifest, cần đăng ký permission được nhận sự kiện này

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

- Khai báo receiver bên trong thẻ application

```
<receiver android:name=".BootReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

Ví dụ BootReceiver

- Ở đây ta khai báo trong manifest là ứng dụng có một receiver tên là BootReceiver
- BootReceiver này đăng ký nhận sự kiện “hệ thống khởi động hoàn tất”.
- Dĩ nhiên, muốn nhận sự kiện dạng này thì cần phải đăng kí trước với hệ thống qua permission để user được biết.

Ví dụ BootReceiver

- Tạo một class mới trong source, tên là BootReceiver kế thừa BroadcastReceiver.
- Implement lại phương thức onReceive()

```
if (intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED)) {
    Intent helloIntent = new Intent(context, HelloBootActivity.class);
    helloIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    context.startActivity(helloIntent);
}
```

Ví dụ BootReceiver

- Giải thích: vì Receiver không kế thừa context nên khi tạo intent mới không truyền this vô được, thay vào đó truyền context đã được gửi kèm.
- Vì không ở trong 1 activity mà đang ở trong 1 receiver, và một số vấn đề liên quan tới task trong Android nên phải thêm cờ Intent.*FLAG_ACTIVITY_NEW_TASK* (chỉ có thể không dùng cờ này khi gọi startActivity() từ một activity)

Ví dụ BootReceiver

- Ứng dụng có một activity tên là HelloBootActivity, activity này sẽ chỉ hiển thị dạng dialog, và sẽ không được start bằng cách bấm vào icon trên màn hình. Vì thế, khai báo trong manifest như sau:

```
<activity android:name=".HelloBootActivity"
 android:theme="@android:style/Theme.Dialog">
</activity>
```

- Còn activity chỉ hiển thị một cái TextView là “Chào bạn, mới khởi động xong” và một Button để bấm vào đó thì đóng activity.

Phát sự kiện

- Có thể phát một sự kiện cho các receiver khác nhận dạng như sau:

```
Intent intent = new
 Intent("org.multiuni.android.BROADCAST_DEMO");
sendBroadcast(intent);
```

- Hoặc:

```
sendOrderedBroadcast(intent, "permission tùy ý hoặc null");
```

- Ngoài ra còn có một số cách gửi broadcast khác, tham khảo thêm trong class ContextWrapper

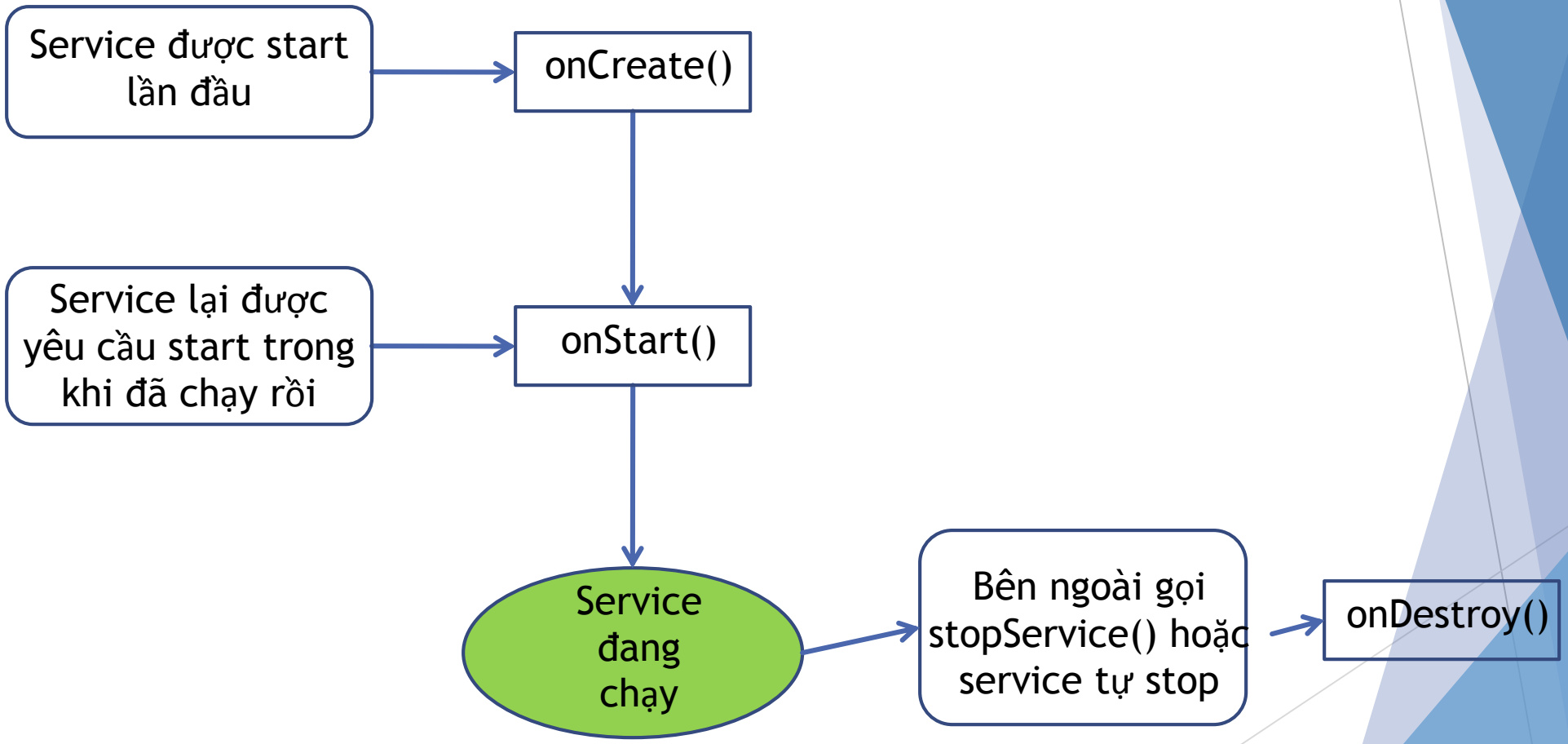
Service

Tổng quan

- Là một trong 4 loại thành phần của một ứng dụng.
- Service chạy nền và không tương tác trực tiếp với người dùng.
- Tham khảo:

http://developer.android.com/reference/android/app/Service.html

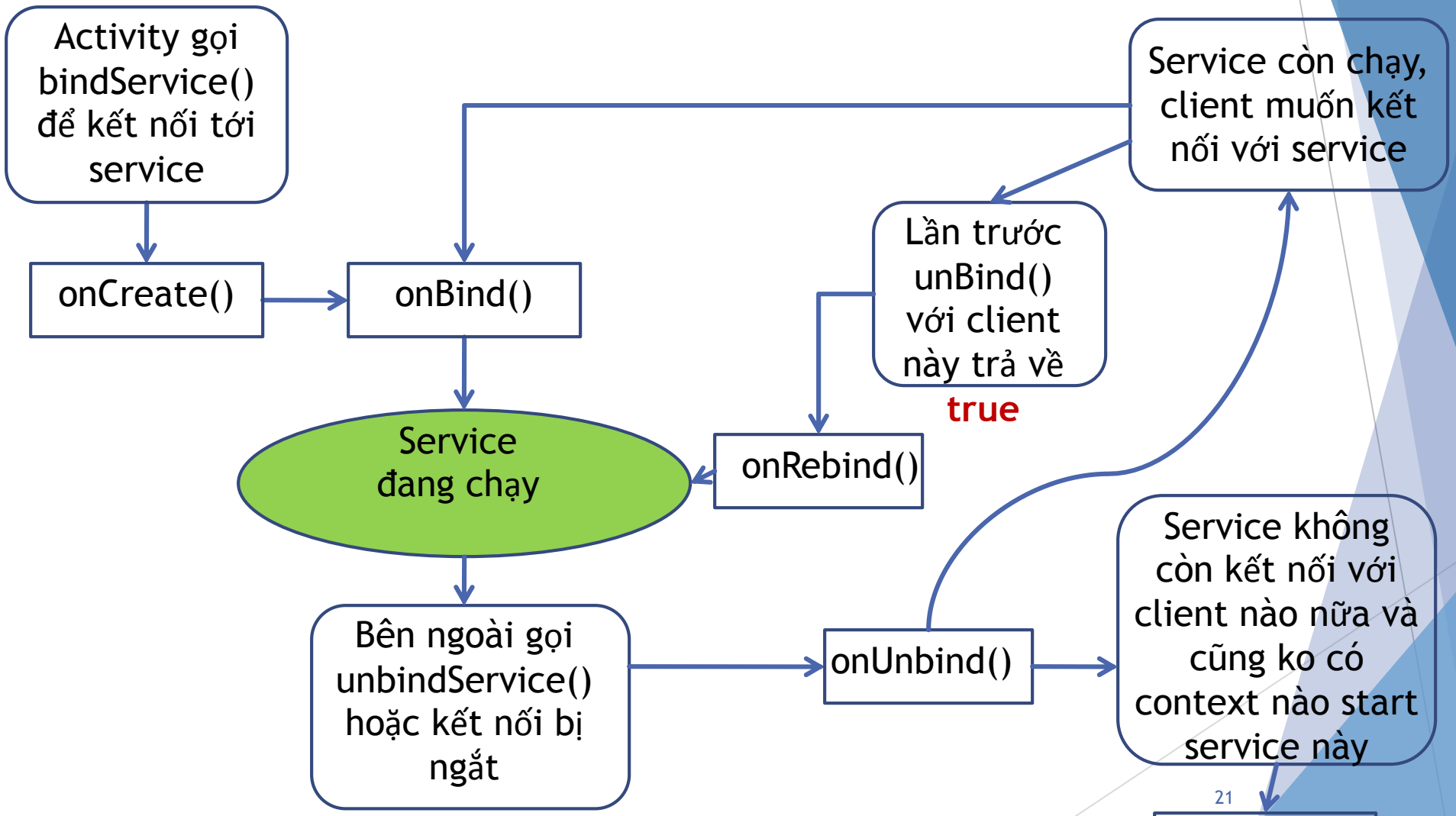
Lifecycle – startService()



Lifecycle – startService()

- Khi có một context nào đó gọi startService() để start service mong muốn. Nếu service đó chưa được tạo thì sẽ gọi onCreate() rồi gọi tiếp onStart() và khi đó service chạy nền bên dưới.
- Nếu sau đó lại có một context muốn start service này mà service đã đang chạy, chỉ có phương thức onStart() của service được gọi.
- Dù service có được gọi start bao nhiêu lần thì cũng chỉ có 1 instance của service và chỉ cần gọi stopService() một lần để kết thúc service.

Lifecycle – bindService()



Lifecycle – bindService()

- Thông thường, lifecycle của service khi có client kết nối từ đầu như sau:
  - Cũng bắt đầu bằng onCreate() rồi đến onBind() và service chạy background.
  - Khi không còn client kết nối tới thì service gọi onUnbind() rồi onDestroy().

Lifecycle

- Có một số trường hợp không thông thường, ví dụ như:
  - Có một context khởi động (start) một service, sau đó có một số client kết nối (bind) tới service
  - Có nhiều client cùng lúc kết nối (bind) tới service
  - Một activity vừa gọi startService() vừa gọi bindService()
  - …

Sử dụng service

- Dùng trong các ứng dụng nghe nhạc.
- Dùng để xử lý các thao tác mất thời gian và không nhất thiết phải hiển thị lên activity (download, upload…)
- Đôi khi cần một ứng dụng vận hành liên tục để xử lý những việc mong muốn mà không làm phiền người dùng → service
- Làm những thao tác tính toán, xử lý đều đặn nào đó và kết quả khi nào người dùng cần thì mới xem

…

Thread

Thread

- Thông thường dùng Thread để xử lý các code xử lý nặng, hoặc mất thời gian có thể gây chậm chương trình hoặc giao diện bị block.
- Thread khá thông dụng và trong Android dùng lớp Thread của Java.
- Mặc định, mỗi ứng dụng chạy trong một process và code được thực thi trong thread chính của process đó.

Thread

- Nếu code xử lý quá lâu, không kịp phản hồi lại các sự kiện người dùng trong 5 giây thì sẽ xuất hiện dialog “Application is not responding” và người dùng có thể force close ứng dụng ngay lập tức.
- Dù không bị force close thì việc ứng dụng bị lag là khó chấp nhận.
- Tham khảo thêm link sau:

  http://developer.android.com/guide/practices/design/responsiveness.html

  http://developer.android.com/guide/practices/design/seamlessness.html

  http://developer.android.com/guide/practices/design/performance.html

Thread

```
Thread thread = new Thread() {
    @Override
    public synchronized void start() {
     // Khởi tạo các đối tượng cần thiết tại đây
     super.start();
    }
    @Override
    public void run() {
     // code xử lý chính của thread trong này
     super.run();
    }
};
thread.start(); //bắt đầu thread
```

Thread

- Lưu ý:
  - Thread lần đầu thực thi gọi phương thức start(), những lần sau chỉ gọi phương thức run(), không gọi start() nữa.
  - Các code xử lý liên quan đến giao diện chỉ được xử lý trong thread chính của ứng dụng (ví dụ load ảnh từ mạng về thì dùng thread, nhưng hiển thị ảnh lên ImageView thì xử lý trong thread chính)
  - Sau khi thực thi xong phương thức run(), thread không còn active nữa.

Handler

- Trong Android, để tiện việc giao tiếp giữa 2 thread ta dùng đối tượng Handler.
- Ngoài ra, có thể dùng Handler để đặt xử lý một yêu cầu nào đó sau một khoảng thời gian xác định.

Handler

- Giao tiếp giữa 2 Thread:
  - Giả sử trong phương thức run() của Thread, đã lấy xong đối tượng Bitmap về. Muốn truyền đối tượng Bitmap cho Thread chính hiển thị lên màn hình:

```
Message msg = mHandler.obtainMessage(1, bitmap);
mHandler.sendMessage(msg);
```

  - Trong code của Activity (mặc định là thread chính), ta khai báo một đối tượng Handler tương tứng như sau:

Handler

```
Handler mHandler = new Handler() {

    @Override
    public void handleMessage(Message msg) {
     if (msg.what == 1) {
            //Hiển thị Bitmap
            mImageView.setImageBitmap((Bitmap)msg.obj);
     }
     super.handleMessage(msg);
    }
};
```

Handler

- Nhờ đối tượng mHandler lấy ra một message và gắn mã vào cho message đó, kèm theo đối tượng bitmap. Sau đó gửi đi.
- Message gửi đi sẽ nhận phương thức callback là handleMessage() của đối tượng Handler.
- Handler còn có thể gửi message để xử lý sau một khoảng thời gian định sẵn sendMessageAtTime hoặc xử lý tại một thời điểm định sẵn sendMessageDelayed … có thể tìm hiểu thêm trong tài liệu của lớp Handler

Handler

- Handler được tạo trong thread nào thì sẽ sử dụng message queue của thread đó.
- Có thể dùng Handler như bộ đếm giây khi chơi nhạc, hoặc chức năng tương tự
- Lưu ý là nếu trong message queue vẫn còn message thì vẫn còn thực thi dù đã thoát khỏi ứng dụng.

AlarmManager

- Dùng AlarmManager để thực hiện đăng ký xử lý một thao tác nào đó tại một thời điểm nhất định trong tương lai (thường là thời gian dài).
- Nếu xử lý trong thời gian ngắn thì khuyến cáo nên dùng Handler.
- Ưu điểm của AlarmManager, khi đến thời điểm được định trước, dù ứng dụng đang không chạy vẫn được gọi.
- Nếu tắt máy thì bật lại cũng không còn (lưu ý điểm này)

AlarmManager

- Khởi tạo một alarm:

```
AlarmManager am = (AlarmManager)   getSystemService(ALARM_SERVICE);

Intent broadcastIntent = new Intent("org.multiuni.android.ACTION...");

PendingIntent pendingIntent = PendingIntent.getBroadcast(this,
    0, broadcastIntent, PendingIntent.FLAG_CANCEL_CURRENT);

am.set(AlarmManager.RTC_WAKEUP, triggerAtTime, pendingIntent);
```

AlarmManager

- Giải thích:
  - Khởi tạo một đối tương AlarmManager để làm việc với Alarm.
  - Tạo một intent tên broadcastIntent, intent này được dùng để gửi broadcast khi đến thời điểm định sẵn.
  - PendingIntent được khởi tạo gồm context, broadcastIntent ở trên và một cờ báo rằng nếu đã có một Alarm tương tự thì bỏ nó đi và dùng cái mới này.

AlarmManager

- Sau cùng, set alarm với 3 thông số:
  - Bộ đếm thời gian (có 4 loại, xem trong document của AlarmManager)
  - Thời gian chính xác để bật alarm lên.
  - PendingIntent gửi đi (dùng để xác định tới thời điểm bật alarm lên thì cần phát intent nào)

Notification

- Trong những trường hợp muốn hiện một thông báo về một sự kiện nào đó cho người dùng mà không muốn ảnh hưởng đến công việc của họ hoặc không chắc họ có đang cầm điện thoại (tin nhắn, cuộc gọi, email…)
- Hoặc muốn hiển thị thông tin một việc nào đó đang xảy ra trên điện thoại và mong người dùng biết (đang nghe nhạc, đang trong cuộc gọi, thiếu thẻ nhớ…)

→ Notification

Notification

- Có thể tạo một notification có âm báo, rung, đèn led, icon…
- Notification có 2 dạng:
  - One time
  - On going

http://developer.android.com/reference/android/app/NotificationManager.html

Q/A