AndroidManifest.xml -> file cần thiết để hệ thống Android có thể chạy được, tất cả cá thành phần có trong một App, ví dụ như mỗi Activity phải được khai báo trong file XML này.

Activity là màn hình hiển thị lên cho người dùng thấy

Add log vào trong code để hiển thị tin nhắn:

```
Log.d("MainActivity", "Hello World");
```

Trong đó

- Log : Log class sử dụng để gửi log message tới Logcat pane
- d : Debug Log, cài đặt để có thể lọc được các message bên cạnh đó còn có e (Error), w (Warn), i (Info)
- "MainActivity" : biến số đầu tiên như là một cái tag sử dụng để có thể lọc được message ở trong Logcat panel
- "Hello world" : biến số thứ 2 như là message thực sự

build.gradle : sử dụng để khi cần thêm các thư viện mới vào trong project
AndroidManifest.xml : thêm tính năng, các thành phần và các quyền cho Android app

Folder java

Chứa các file Java. Mỗi Activity, Service hoặc là một thành phần (ví dụ như Fragment) đều được định nghĩa như là một Java class

Layout files

Để xem và chỉnh sửa một layout file, mở folder res, vào folder layout, ở ví dụ trên, đối với MainActivity.java có file layout là activity_main.xml

Resource file

Folder res chứ các resources ví dụ như layouts, string, images. Một Activity luôn luôn có quan hệ với layout của một UI được định nghĩa bằng các file XML. File XML này thường được đăt tên phía sau Activity. Thư mục res chứ các thư mục con sau:

- drawable : Chứa các ảnh có bên trong app
- layout : Với mỗi Activity có ít nhất một file layout viết bằng XML để có thể miêu tả UI
- mipmap : Sử dụng để có thể chứa icon khởi tạo của App. Trong này có một thư mục con dành cho mỗi mật độ điểm ảnh được hỗ trợ. Android sử dụng mật độ điểm ảnh để có thể quyết định độ phân giải cần thiết của hình ảnh
- values : thư mục này chứa các định nghĩa về string, màu, các miền để tránh hardcode bên trong file XML và file Java
    - colors.xml : sử dụng để định nghĩa các màu cần thiết
    - demens.xml : sử dụng để định nghĩa các size của view và các đối tượng dành cho các đọ phân giải khác nhau
    - strings.xml : sử dụng để làm localize
    - styles.xml : Theme và styles của map đều đến từ đây. Styles giúp app có thể trông thống nhất hơn với tất cả các thành phần Ui

Android manifest

Trước khi hệ thống Android có thể bắt đầu các thành phần của App như là một Activity, hệ thống phải biết được các Activitiy đang tồn tại. Làm được việc đó bằng cách đọc file Android Manifest.xml của app, cái này sẽ mô tả các thành phần của một app Android. Mỗi Activity pahri được liệ kê trong file XML này cùng với các thành phần dành cho App\

Các layout và resource dành cho UI

Views

UI bao gồm một hệ thống phân cấp các đối tượng gọi là views - mỗi thành phần của màn hình là một View. Lớp View biển diễn cho một khối xây dựng

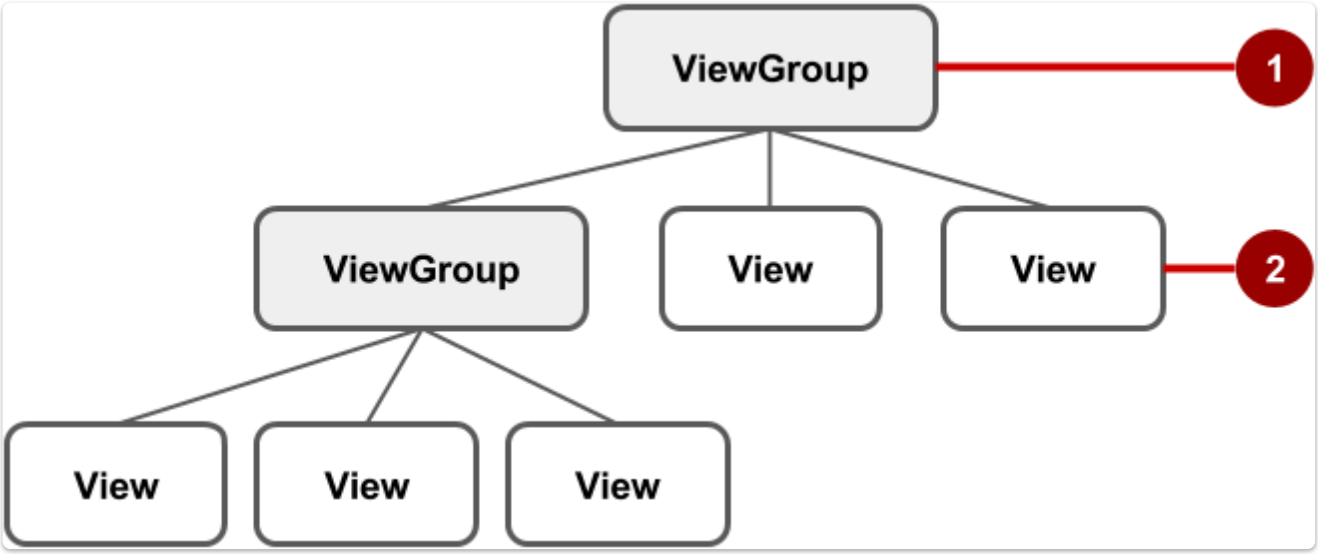
cơ bản cho tất cả các thành phần UI, và lớp cơ sở cho các class cung cấp các thành phần UI tương tác như là button, checkboxes và text

ViewGroup groups

Các thành phần View được nhóm lại bên trong một ViewGroup, thứ mà hoạt động như là một container. Mối quan hệ giữa chúng là cha-con, trong đó cha là ViewGroup và con là View và các ViewGroup khác.

Bố cục nhóm các ViewGroup

Các thành phần View dành cho một màn hình được tổ chức thành một hệ thống phân cấp. Tại rôt của hệ thống phân cấp này là một ViewGroup chứa bố cục của toàn bộ màn hình. ViewGroup này chứ các thành phần View con và nhưng ViewGroup khác như hình được show dưới đây



Một vài ViewGroup được thiết kế như là bố cục bởi vì chúng tổ chức các thành phần View con theo một các riêng và thường được sử dụng làm ViewGroup gốc

Sử dụng ConstraintLayout

Một *constraint* là một kết nối / một sự điều chỉnh tới thành phần UI khcs, tới parent của bố cục haowcj là một hướng dẫn vô hình. Mỗi constraint xuất hiện như là một dòng mởi trộng từ một tay nắm tròn

- Baseline Constraint : căn chỉnh một thành phần UI chứa text chẳng hạn như TextView hoặc là Button với một thành phần UI khác có chứa text.

Một *baseline constraint* cho phép bạn constraint các thành phần để từ đó text baseline match được với nhau

- match_constraint : sử dụng để fill hết toàn bộ bố mẹ cho tới lề (nếu như lề được set)
- wrap_content : thu nhỏ thành phần UI bằng với kích thước của nội chung chứa bên trong nó, nếu như không có nội dung -> không thể nhìn thấy
- Sự dụng số cự thể nếu như muốn set chiều dài / rộng

```
android:id="@+id/button_count"
sử dụng để định danh một thành phần View nào đó
sử dụng dấu + để biểu thị rằng bạn tạo mới một id
bỏ dấu + đi khi muốn refer đến một View nào đó
```

Sử dụng LinearLayout

> *Là một layout tuyến tính giống như Row và Column*
> *LineaerLayout yêu cầu các thuộc tính sau phải được set*
>
> - *android:layout_width : chiều rộng*
> - *android:layout_height : chiều dài*
> - *android:orientation : hướng của layout*
> - *android:layout_gravity : quyết định cách sắp xếp các thành phần so với bố mẹ của nó*
> - *android:padding*
> - *andoird:padding(T/B/L/R)*
> - *android:paddingStart : Padding ở cạnh bắt đầu của View*
> - *android:paddingEnd : Padding ở cạnh kết thúc của View*

Sử dụng RelativeLayout

> *Layout này làm cho các con của nó có mối quan hệ với nhau, sử dụng một element khác để làm mốc thiết lập vị trí cho một element mới, các thuộc tính mà các con ở bên trong RelativeLayout có thể có*
>
> - *android:layout_toLeftOf : bên trái của một bên cách bên phải của một bên khác*

> - *android:layout_toRightOf : bên phải của một bên so với bên trái của một bên khác*
> - *android:layout_centerHorizontal : căn giữa theo chiều ngang của bố mẹ*
> - *android:layout_centerVertical : căn giữa theo chiều dọc của bố mẹ*
> - *android:layout_alignParentTop : căn theo top của bố mẹ*
> - *android:layout_alignParentBottom : căn theo bottom của bố mẹ*

TextView

> *Sử dụng để hiển thị text lên màn hình*
> *Các thuộc tính có thể*
> *android:text : sử dụng để set text hiển thị*
> *android:textColor : sử dụng để set màu hiển thị text*
> *android:textAppearance : sử dụng để link tới một resource khác*
> *android:textSize : sử dụng để set text size (đơn vị là sp - scaled-pixel)*
> *android:typeface : sử dụng để set kiểu chữ*
> *android:lineSpacingExtra : sử dụng để set khoảng cách giữa các dòng text*
> *android:autoLink : sử dụng để điều kiển text như là một link url / email*

ScrollView

Class cung cấp bố cục cho các view scroll theo chiều dọc, là một lớp con của FrameLayout
Tất cả nội dung của một ScrollView chiếm bộ nhớ kể cả khi vị trí của chúng có được hiển thị hay không. Điều này làm cho ScrollView hữu dụng cho các page muốn scroll mượt và có text bởi vì text đã rẵn sàng ở trong bộ nhớ. Tuy nhiên ScrollView với một ViewGroup có chứa nhiều thành phần View sẽ chiếm nhiều bộ nhớ, có thể ảnh hưởng tới hiệu năng của toàn bộ app

Activity

Một *activity* biểu diễn cho một màn hình riêng lẻ trong app với giao diện người dùng có thể tương tác được.
Mỗi khi một *activity* mới được bắt đầu, *activity* trước đó sẽ bị dừng tại, nhưng hệ thống giữ hoạt đọng đó trong một stack. Khi người dùng xong với hoạt

động hiện tại và bâm snuts quay lại, họt động sẽ được lấy ra từ stack và hủy, và hoạt động trước đó sẽ được tiếp tục

Khi một *activity* bị dừng lại bởi vì một hoạt động moiwst bát đầu, hoạt động đầu tiên được thông báo bắng các lifecycle method - tập các trạng thái của một *Activity* có thể tồn tại : khi một hoạt động được khởi tạo lần đầu tiên, khi chúng dừng, khi chún dược tiếp tục và khi cả hệ thống bị phá hủy

Tạo Activity trong App

> - *Tạo một Activity Java class*
> - *Triển khai UI của Activity trong một file layout XML*
> - *Định nghĩa hoạt động mới bên trong AndroidManifest.xml*
> - *Triển khai các phương thức vòng đời cần thiết và cái quan trọng nhất là **onCreate()***
> - *Bên trong **onCreate()** có gọi tới **setContentView()** để thực hiện khởi tạo layout UI*

Lưu ý khi mới khởi tạo Activity

Khi bạn tạo một project mới trong Android Studio và chọn Backwards Compatibility, MainActivity được mặc định , đó là một lớp của AppCompatActivity. Lớp AppCompatActivity cho phép bạn sử dụng các tính năng mới nhất của Andrỏid như là app bar và Material Design trong khi vẫn có khả năng máy của bạn tương thích với các thiết bị chạy các Android version cũ hơn

Nhiệm vụ đầu tiên của bạn trong một lớp con *Activity* là triển khai phương thức vòng đời tiêu chuẩn để có thể handle được các trạng thái thay đổi với *Activity* của bạn. Những trạng thái thay đổi này bao gồm những thứ ví dụ như khi *Activity* được khởi tạo, dùng lại, tiếp tục và phá hủy

Một callback yêu cầu mà app của bạn bắt buộc phải triển khai đó là phương thức onCreate(). Hệ thống gọi method này khi nó khởi tạo *Activity* của bạn và tất cả các thành phần quan trong trong *Activity* của bạn đều được khởi tạo ở đây. Quan trong nhất, phương thức *onCreate()* gọi *setContentView()* để khởi tạo layout tiêu chuẩn cho *Activity*

Layout và Activity

Bạn thường định nghĩa UI dành cho *Activity* trong một hoặc nhiều file bố cục XML. Khi phương thức **setContentView()** được gọi với đường dẫn tới một file layout, hệ thống tạo ra tất cả các view ban đầu từ các layout riêng biệt và thêm chúng vào *Activity* của bạn. Như này thường được gọi là inflating layout

Triển khai hoạt động của UI

UI dành cho một UI cung cấp một hệ thống phân cấp các thành phần View, thú điều khiển một khoảng không gian cụ thể bên trong cửa sổ activity và có thể phản hồi với tương tác của người dùng.

Các thông thường nhất để định nghĩa một UI sử dụng thành phần View là: với một file bố cục XML lưu trữ như là một phần tài nguyên app của bạn. Định nghĩa bố cục của ban trong XML cho phép bạn có thể duy trì thiết kế của UI độc lập với source code - nơi định nghĩa các hành vi của *Activity*

Khai báo Activity trong AndroidManifest.xml

Với mỗi Activity trong app của bạnh phải được khởi tạo ở trong Androidmanifest.xml với thành phần activity, bên trong khu application. Khi bạn tạo một project mới haowcj thêm ột hoạt động mới vào project trong Android Studio, AndroidManifest.xml đươc tạo hoặc update để bao gồm khung khởi tạo cho mỗi Activity. Dưới đây một khai bào dành cho **MainActivity**

```
<activity android:name=".MainActivity">
        <intent-filter>
                <action
android:name="android.intent.action.Main"/>
                <category
android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
</activity>
```

Thành phần chữa một só các thuộc tính để định nghĩa các properties của *Activity* như là label, icon, hoặc là theme. Thuộc tính yêu cầu bát buộc duy

nhất đó chính là *android:name*, chỉ rõ tên class của *Activity*.

Thành phần có thể cũng bao gồm khai báo dành cho bộ lọc *Intent*. Bộ lọc *Intent* chỉ rõ loại *Intent* mà *Activity* sẽ chấp nhận

Intent filter bắt buộc phải bao gồm ít nhất một element, và bạn có thể cũng bao gồm một hoặc một lựa chọn . MainActivity dành cho app của bạn cần một bộ lọc *Intent* ddeer định nghĩa hoạt động "chính" và danh mục khởi tạo và từ đó hệ thống có thể thực hiện launch app của bạn. Android Stuido tạo *Intent* filter này cho *MainActivity* trong project của bạn

- element chỉ ra nó là một entry point chính của app
- element chỉ ra rằng hoạt ododngj này nên được liệt kê vào trình khởi tạo của hệ thống (để cho phép ngườ dùng chạy hoạt động này)
  Mỗi *Activity* trong app của bạn cũng cần được khai báo *Intent* filter nhưng chỉ *MainActivity* nên khai báo là "main"

Intents

Mỗi hoạt động được bắt đầu hoặc được kích hoạt với một *Intent* - là một đối tượng thông điệp sẽ tạo một yêu cầu tới Android runtim để bắt đầu một hoạt động hoặc những thành phần khác của app hoặc trong các app khác

Khi app của bạn lần đầu tiên được chạy từ màn hình chính của thiết bị, Android runtime gửi một *Inten* tới app của bạn để bắt đầu hoạt động chính của app (cái mà được định nghĩa mà **MAIN** action và có thể loại là **LAUNCHER** bên trong file **AndroidManifest.xml**) . Để bắt đầu các hoạt động khác trong app cụa bạn, hoặc để yêu cầu một vài hoạt động khả dụng trên thiết bị cảu bạn thực hiện một hành động, bạn có thể xây dựng intent riêng của bạn và gọi phương thức **startActivity()** để gửi intent

Thêm vào đó, để bắt đầu một hoạt động, một mục tiêu có thể sử dụng để có thể pass data giữa các hoạt động này và hoạt động khác. Khi bạn tạo một intent để bắt đầu một hoạt động mới, bạn cần bao gồm thông tin về dữ liệu mà bạn muốn hoạt động mới để có thể thực hiện. Ví dụ, một email *Activity* hiển thị danh sách các thông điệu có thể gửi một *Intent* tới *Activity* mà hiển thị message. Hiển thị *Activity* cần thông tin về message đó để hiển thị, và bạn có thể chứa data vào bên trong intent.

Intent type

Intents có thể tường minh hoặc không tường minh

- Intent tường minh: Bạn chỉ định hoạt động nhận bawgf cách sử dụng tên lớp điều kiện của hoạt động. Bạn sử dụng *Intent* tường minh để bắt đầu app của chính bạn (ví dụ, để di chuyển giữa 2 màn hình trong UI) bở vì bạn đã biết package và lớp class của component đó
- Intent không tường minh: Bạn không chỉ rõ một hành động cụ thể nào hay component nào nhận intent. Thay vào đó bạn khai báo một hành động chung để xảy ra và hệ thống Android kết nối yêu cầu của bạn tới một *Activity* hoặc một component kahcs mà có thể handle được hành động vừa được request. Bạn học về sử dụng Intent khong tường minh trong các thực hành khác

Intent object và các trường

Với một *Intent* tường minh, các trường chính bao gồm:

- Lớp *Activity*(**class**). Đây là tên lớp của *Activity* hoặc thành phần khác nên nhận *Intent*; ví dụ, com.example.SampleActivity.class. Sử dụng *Intent* constructor hoặc *setComponent()*, *setComponentName()*, hoặc *setClassName()* để chỉ định lớp
- Dữ liệu Intent (**data**). Trường dữ liệu chứa một tham chiếu tới dữ liệu bạn muốn Activity nhận và xử lý như là một Uri object
- Intent mở rộng (**extras**). Những cặp key-value mang thông tin mà Activity nhận yêu cầu để hoàn thành hành động được yêu cầu
- Itent **flags** : Những bit thêm vào của metadata, được định nghĩa bởi *Intent* class. Các cờ này có thể hướng dẫn cho hệ thông Android cách để có thể khở tạo một *Activity* và các để có thể đối xử với nó sau khi nó đã được khởi chạy

Với một *Intent* tường minh, bạn có thể cần phải định nghĩa hành động *Intent* và category.

Bắt đầu một Activity với *Intent* tường minh

Để bắt đàu một Activity cụ thể từ một Activity kahcs, sử dụng *Intent* tường minh và phương thức *startActivity()*. Một *Intent* tường minh bao gồm tên đầy đủ của lớp dành cho *Activity* hoặc thành phần khác trong một đối tượng *Intent*. Tất cả các trường khác của *Intent* đều là tùy chọn, null là gì trị default

Với ví dụ, nếu bạn muốn bắt đầu hoạt động *ShowMessageActivity* để hiển thị tin nhắn chỉ định trong một email app, sử dụng code như sau

```
Intent messageIntent = new Intent(this,
ShowMessageActivity.class);
startActivity(messageIntent);
```

Hàm khởi tạo của *Intent* lấy vào 2 tham số đối với *Intent* tường minh

- Một *application content*, trong ví dụ này, lớp *Activity* cung cấp context(this)
- Thành phần chỉ định để bắt đầu (*ShowMessageActivity.class*)

Sử dụng phương thức startActivity() với một đối tượng Intent mới như là tham số duy nhất. Phương thức *startActivity()* gửi một *Intent* tới hệ thống Android, khi khởi chạy lớp *ShowMessageActivity* trong thay mặt cho app cảu bạn. Một hoạt động mới xuất hiện tren màn hình, và hoạt động khởi nguồn bị dừng lại

Activity được khởi tạo còn lại trên màn hình cho tới khi người dùng tap vào nút Back trên thiết bị, tại chính thời didemr này, *Activity* đóng lại và được thu giữ lại bởi hệ thống, và *Activity* khởi nguồn được tiếp. Bạn có thể cũng đóng luôn *Activity* được khởi tạo trong phản hồi của người dùng (ví dụ như một Button clcik) với finish() method

```
public void closeAcivity(View view) {
        finish();
}
```

Truyền dữ liệu từ một Activity tới một Activity khác

Thêm vào đó, để đơn giản khởi tạo một *Activity* từ một *Activity* khác, bạn cũng có thể sử dụng một *Intent* để truyền thông tin từ một *Acitivity* tới một

Activity khác. Đối tượng *Intent* bạn sử dụng để chạy *Activity* có thể chứa *Intent* datas hoặc là *Intent* extras - những bit dữ liệu thêm vào mà *Activity* có thể cần
Trong *Activity* đầu tiên (gửi đi), bạn:

1. Tạo một đối tượng *Intent*
2. Đẩy dữ liệu / extras vào bên trong *Intent*
3. Bắt đầu một *Activity* mới bằng *startActivity()*
   Trong *Activity* thứ hai, bạn:
4. Nhận đối tượng *Intent* và bắt đầu
5. Khôi phục dữ liệu / extra từ đối tượng *Intent*

Khi sử dụng Inten data hoặc là Intent extras

Bạn có thể sử dụng cả *Intent* dât hoặc *Intent* extra để truyền dữ liệu từ một *Activity* với một *Activity* khác. Có khá nhiều khác nhau giữa data và extra có thể quyết định bạn nên sử dụng cái nào

Intent data có thể chứa chỉ một mảnh của thông tin : một URI biểu diễn nơi mà data bạn muốn vận hành. ỦI có thể là một web page URL, một số điện thoại, một địa chỉ vật lý hoăc bất kì URI nào bạn định nghĩa

Sử dụng trường *Intent* data:

- Khi bạn chỉ có một mẩu thông tin và bạn muốn gửi tới *Activity* được bắt đầu
- Khi mà thông tin là một vị trí data và có thể được biểu diễn bởi một URI

Intent extra được dành cho bất kì dữ liệu tùy ý nào khác mà bạn muốn chuyển đến hoạt động được bắt đầu. *Intent* extra được lưu trữ như làm một đối tượng *Bundle* như là các căp key và value. Một *Bundle* mà một map, tối ưu dành cho Android, với key mà một stirng, và một giá trị có thể là bất kì kiểu nguyên thủy hoặc kiểu đối tượng (đối tượng phải được triển khai từ *Parcelable* interface). Để có thể đẩy data vào trong Intent extras bạn có thể sử dụng bất kì phương thức *putExtra()* của bất kì lớp *Intent* nào, hoặc bạn có thể tạo một *Bundle* của chính bạn và đẩy nó vào bên trong *Intent* với *putExtras()*

Sử dụng *Intent* extras:

- Nếu bạn muốn truyền nhiều hơn một mẩu thông tin tới *Activity* được bắt đầu
- Bát kì thông tin nào bạn muốn pass mà không được biểu thị thông qua một URI
  Intent data và extras chúng không độc lập, bạn có thể sử dụng dữ liệu cho một URI và extra dành cho bất kì thông tin thêm nào và mà *Activity* được bắt đâuf muốn để xử lý thông tin đó trong URL

Thêm dữ liệu vào *Intent*

Để có thể thêm dữ liệu vào *Intent* tường mình từ *Activity* khởi nguồn, tạo mọt đối tượng *Intent* như bạn đã làm trước đây:

```
Intent messageIntent = new Intent(this,
ShowMessageActivity.class);
```

Sử dụng phương thức *setData()* với một đối tượng Uri để thêm một URI vào *Intent*. Một vài ví ụ dưới đây sử dung *setData()* với URIs:

```
messageIntent.setData(Uri.parse("http://goole.com"));
messageIntent.setData(new File("/sdcard/sample.jpg"));
messageIntent.setData(Uri.parse("content://mysample.provider/d
ataa"));
```

Luôn chú ý rằng trường dữ liệu có thể chỉ chứa một URI đơn lẻ, nêu bạn muốn gọi *setData()* nhiều lần chỉ lần cuối cùng được sử dụng. Sủ dụng *Intent* extras để chứa các thông tin thêm vào (bao gồm cả URI)
Sau khi bạn thêm dữ liệu, bạn cần bắt đầu Activity với Intent như bình thường:

```
startActivity(messageIntent);
```

Thêm extras vào *Intent*

Để thêm *Intent* extras vào một *Intent* tường mình từ *Activity* khởi nguồn

1. Xác định khóa sử dụng cho thông tin mà bạn muốn để vào trong extras, hoặc là bạn tự định nghĩa. Mỗi mảnh thông tin cần một khóa duy nhất

2. Sử dụng phuowng thức *putExtra()* để thêm cặp key/value vào *Intent* extras. Tùy thuộc bạn có thể sử dụng đối tượng *Bundle*, thêm dữ liệu của bạn vào *Bundle* và sau đó thêm *Bundle* vào *Intent*

Lớp *Intent* chứa cá khóa mở rộng mà bạn có thể sử dụng, định nghĩa như alf một hằng, bắt đầu bằng từ **EXTRA_**. Ví dụ, bạn có thể sử dụng *Intent.EXTRA_EMAIL* để bieeiur thị cho mảng các địa chỉ email, haowcj là *Intent.EXTRA_REFERER* để chỉ rõ thông tin về *Activity* khởi nguồn đã gửi *Intent*

Bạn có thể định nghĩa khóa mở rộng cho *Intent*. Thông thường bạn định nghĩa khóa mở rộng cảu *Intent* như là một biến static với tên bắt đầu bằng *EXTRA_*. Để chắc chăn nó là duy nhất, giá trị string từ khóa của nó nên có tiền tồ với full name app của bạn, ví dụ dưới đây

```
public final static String EXTRA_MESSAGE =
"com.example.mysampleapp.MESSAGE";
public final static String EXTRA_POSITION_X =
"com.example.mysampleapp.X";
public final static String EXTRA_POSITION_Y =
"com.example.mysampleapp.Y";
```

Tạo một đối tượng *Intent*

```
Intent messageIntent = new Intent(this,
ShowMessageActivity.class);
```

Sử dụng phương thức *putExtra()* với một key và để dữu liệu vào trong *Intent* extras. Lớp *Intetn* định nghãi nhiều phương thức *putExtra()* dành cho nhiều loại data:

```
messageIntent.putExtra(EXTRA_MESSAGE, "this is my message");
messageIntent.putExtra(EXTRA_POSITION_X, 100);
messageIntent.putExtra(EXTRA_POSITION_Y, 500);
```

Hoặc là bạn có thể tạo mới một *Bundle* và cho *Bundle* đó vào *Intent* extras. *bundle* định ngahix nhiều hmaf puts dành cho các loại data nguyên thủy khác nhau tốt như đối tượng được triển khai bằng *Parcelable* interface hoặc là Java's *Serializable*

```
Bundle extras = new Bundle();
extras.putString(EXTRA_MESSAGE, "this is my message");
extras.putInt(EXTRA_POSITION_X, 100);
extras.putInt(EXTRA_POSITION_Y, 500);
```

Khôi phục dữ liệu từ *Intent* trong *Activity*

Khi bạn bắt đầu một Activity với một *Intent*, đối tượng được bắt đầu truy cập vào *Intent* vaf dữ liệu nó chứa
Để có thể phục hồi *Intent* sử dụng phương thức *getIntent()*

```
Intent intent = getIntent();
```

Sử dụng *getData()* để lấy URI từ *Intent*:

```
Uri locationUri = intent.getData();
```

Để có thể lấy được extras bên ngoài *Intent*, bạn cần biết keys cho cặp key/value. Bạn có thể sử dụng extras tiêu chuẩn nếu như bạn sử dụng chúng, hoặc bạn có thể sử dụng khóa mà bạn đã định nghĩa tại *Activity* khởi nđầu
Sủ dụng *getExtra()* để có thể trích xuất dữ liệu bên ngoài *Intent*:

```
String message =
intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
int positionX =
intent.getIntExtra(MainActivity.EXTRA_POSITION_X);
int positionY =
intent.getIntExtra(MainActivity.EXTRA_POSITION_Y);
```

Lấy data lại từ một Activity

Khi bạn bắt đầu một *Activity* với một *Intent*, *Activity* khởi đầu sẽ bị tạm dừng, và *Activity* mới còn lại trên màn hình cho tới khi người dùng click vào nút Back, hoặc bạn gọi phương thức *finish()* trong một click handler hoặc một hàm khác và kết thúc sự tham gia của người dùng với *Activity*

Thỉnh thoảng khi bạn gửi dữ liệu từ một *Activity* với một *Intent*, bạn muốn lấy dữ liệu lại gừ *Intent* đó. Ví dụ, bạn có thể bắt đầu thư viện ảnh *Activity* và cho ngườ dùng chọn một bức ảnh. Trong trường hợp *Activity* nguyên thủy cần

nhận được thông tin về bức ảnh mà người dùng chọn từ *Activity* được khởi chạy

Để chạy một *Activity* mới và nhận két quả trả về, thực hiện các bước sau trong *Activity* nguyên thủy

1. Thay vì chạy *Activity* với *startActivity()*, gọi hàm *startActivityForResult()* với *Intent* và một request code
2. Tạo một Intent mới trong *Activity* được khởi chạy và thêm dữ liệu trả về vào *Intent* đó
3. Triển khia phương thức *onActivityResult()* ở *Activity* nguyên thủy để xử lý dữ liueej từ data trả về

Sử dụng startActivityForResult() để khởi chạy Activity

Để lấy được dữ liệu lại từ *Activity* được khởi chạy, bắt đàu *Activity* với *startActivityForResult()* thay vì sử dụng *startActivity()*

```
startActivityForResult(messageIntent, TEXT_REQUEST);
```

Phương thức *startActivityForResult()* giống như *startActivity()* lấy một *Intent* chứ thông tin về *Activity* được khởi chạy và bất kì dữ liệu nào được gửi đến *Activity* đó. Tuy nhiên nó cũng cần một request code.
Request code là một số tự nhiên định xác định cho yêu cầu và có thể sử dụng phân biệt giữa các kết quả khi bạn muốn xử lý dữ liệu trả về. Ví dụ, nếu bạn chạy một Activity và lấy mọt ảnh và một ảnh khác để lấy một ảnh từ thư viện ảnh, bạn cần các request code khác nhau để xác định cái nào data trả về thuộc về
Thông thường bạn định nghĩa các request code như là các biến integer static với tên của nó bao gồm REQUEST sử dụng các số tự nhiên khác nhau cho mỗi code

```
public static final int PHOTO_REQUEST = 1;
public static final int PHOTO_PICK_REQUEST = 2;
public static final int TEXT_REQUEST = 3
```

Trả về một phản hồi từ *Activity* được khởi chạy

Phản hồi dữ liệu từ một *Activity* được khởi chạy trở lại *Activity* khởi đầu được gửi trong một *Intent*, cũng như vậy trong data hoặc extras. Bạn có thể cấu trúc *Intent* trả về này và để dữ liệu vào trong nó giống như cách mà bạ gửi *Intent*. Tiêu biểu *Activity* được khởi tạo sẽ có một *onClick* hoặc là những phuong thức được gọi khác khi bạn xử lý hành động của người dùng và đóng *Activity*. Đây cũng là nơi mà bạn sẽ cấu trúc sự phản hồi
Để có thể trả lại dữ liệu từ *Activity* được khởi chạy, tạo một đối tượng *Intent* mới

```
Intent returnIntent = new Intent();
```

Kết quả *Intent* không một lớp / compoent để kết thúc ở đúng vị trí. Hệ thống Android định hướng phản hồi này tới *Activity* khởi đầu cho bạn. Thêm data/extras vào *Intent* giống như cách bạn đã làm với *Intent* khởi đầu. Bạn có thể cần định nghĩa thêm keys dành cho *Intent* extras trả về tại lớp mà bạn bắt đầu

```
public final static String EXTRA_RETURN_MESSAGE =

"com.example.mysampleapp.RETURN_MESSAGE";
```

Sử dụng phương thức *setResult()* với response code và *Inetnt* với dữ liệu trả về

```
setResult(RESULT_OK, replyIntent);
```

Response code được định nghĩa bởi lớp *Activity* có thể là:

- *RESULT_OK* : request thành công
- *RESULT_CANCELED* : request bị hủy bỏ
- *RESUT_FISRT_USER* : dành cho code bạn tự định nghĩa
  Cuối cùng gọi *finish()* để đóng *Activity* và tiếp tục *Activity* khởi đầu

```
finish()
```

Đọc dữ liệu phản hồi trong onActivityResult()

Hiện tại, *Activity* được khởi tạo đã gửi dữ liệu lại cho *Activity* khởi đầu với một *Intent*, *Activity* đầu tiên phải handle dữ liêu jđó. Để có thể handle dữ liệu trả về bên trong *Activity* khởi đầu, triển khai phương thức *onActivityResult()*

```
public void onActivityResult(int requestCode, int resultCode,
Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == TEXT_REQUEST) {
        if (resultCode == RESULT_OK) {
            String reply =

data.getStringExtra(SecondActivity.EXTRA_RETURN_MESSAGE);
                // process data
        }
    }
}
```

Có 3 tham số ben trong *onActivityResult()* chứ tất cả thông in mà bạn cần để có thể hanle dữ liệu trả về

- Request code: request coed bạn đã thiết lập khi bạn khởi chạy với *Acitivity* với *startActivityForResult()*. Nếu bạn khởi chạy một *Activity* khác để thực hiện các hành động khác nhau, sử dụng code này để xác định dữ liệu cụ thể bạn sẽ lấy lại
- Result code: Result code được set bởi *Activity* được khởi chạy, thường là một trong *RESULT_OK* hoặc *RESULT_CANCLED*
- Intent data: *Intent* chứa dữ liệu trả về

Activity navigation

Hệ thống Android Navigation hỗ trợ 2 cách thức điều hướng khác nhau cho app của bạn

- Back navigation, được cũng cấp bởi nút Back triên thiết bị
- Up navigation, cungcapas bởi bạn như là một tùy chọn trên app bar

Back navigation, tasks, và back stack

Back navigation cho phép người dùng của bạn trở lại một *Activity* trước đó bằng cách chạm vào nút *Back*.
Back stack là một tập hợp của từng hoạt động mà người dùng đã truy cập và có thể trở lị sử dụng back button

Android cung cấp *back stack* dành cho mỗi task. Một task là một organizing concep dành cho mỗi *Activity* người dùng tương tác với khi thực hiện một thao tác, bất kể chung ở trong app của bạn hay là giao với nhiều app khác. Hầu như các task bắt đầu từ Home screen và chạm vào một app icon để bắt đầu một *task* (và một *back stack* mới). Nếu người dùng đang sử dụng 1 app, tab home, và chạy mới app mới, app mới sẽ được chạy trên task và back stack của riêng nó. Nếu người dùng trở lại app ban đầu, back stack và task ban đầu sẽ được trả lại. Sử dụng *Back button* chỉ có thể trả lại các *Activity* trong task hiện tại, không phải tất cả các *task* đang chạy trên thiết bị. Android cho phép người dùng di chuyển giữa các task với nhau bằng một overview / task screen

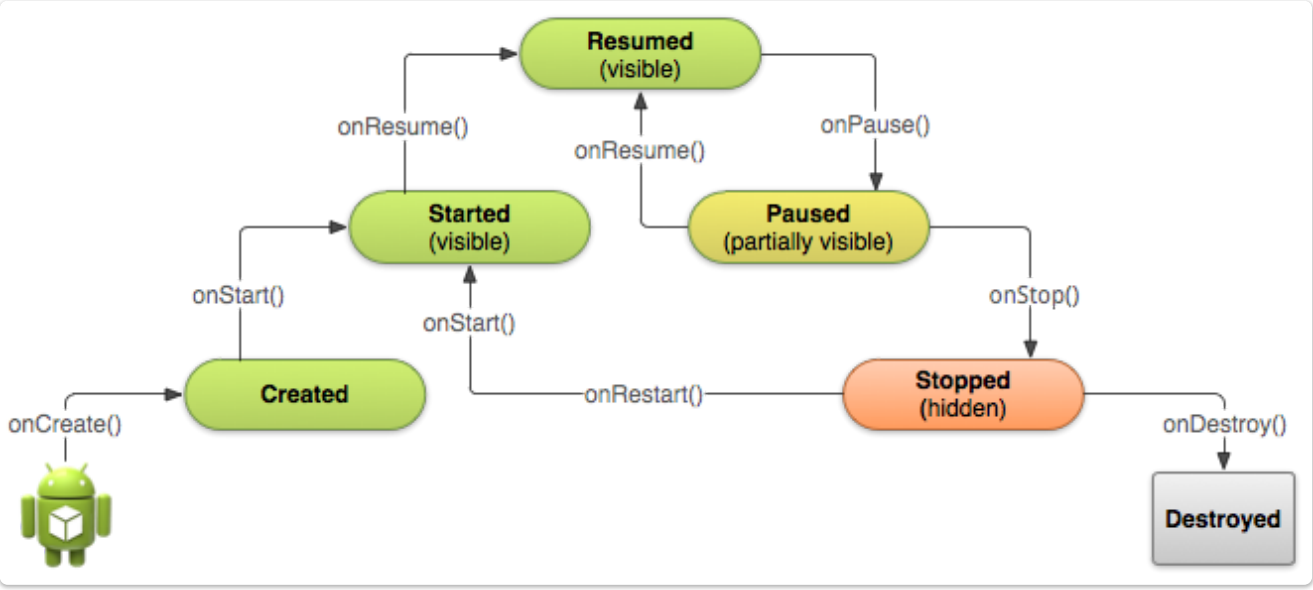
Up navigation

Up navigation, thỉnh thoảng được xem như là tổ tiên navigation / logical navigation, được sử dụng để diu chuyển bên trong một app dựa trên mối quan hệ phân cấp tường minh giữa các màn hình. Với up navigation, mỗi *Activity* được sắp xếp trong một phân cấp, mỗi *Activity* con được hiển thị một mũi tên trải trên app bar để biểu thị rằng trở về *Activity* cha. Cao nhất trong cây phân cấp này là *MainActivity* nên người dùng không thể tiếp tục ở đây
Ví dụ, nếu như *Activity* chính trong một email app là liệt kê tất cả các tin nhắn, chọn một tin nhắn và chạy *Activity* thứ hai để hiển thị email đơn lẻ đó. Trong trường hợp tin nhắn *Activity* cung cấp một Up button để trở lại list các tin nhắn
Hành vi của Up button được đinh nghĩa bởi bạn trong mỗi *Activity* dựa trên các bạn thiết kế app navigation như nào. Trong nhiều trường hợp, Up và Back navigation có thể cung cấp cùng chung một hành vi: để chỉ trở lại *Activity* trước.

Actitity lifecycle và trạng thái

Về vòng đời của Activity

Vòng đời hoạt động là tập các trạng thái của một hoạt đọng có thể có trong suốt toàn bộ quãng đời, từ thời điểm nó được khởi tạo cho tới khi nó bị phá hủy bởi hệ thống. Như là một người dùng tương tác với app của bạ, và app khác trên thiết bị, các hoạt động di chuyển và các trạng thái khác nhau



Trạng thía và các phương thức vòng đời

Khi một *Activity* chuyển và thoạt ra khỏi các trạng thai vòng đời khác nhau, hệ thống Android gọi các phương thức vòng đời tại mỗi trạng thái. Tất cả các phương thức được móc nối và bạn có thể ghi đè trong mỗi lớp *Activity* để định nghĩa *Activity* sẽ phản ứng khi người dùng ra/vào lại *Activity*. Giữa trong đầu rằng các trạng thái vòng đời -> chỉ có *Activity*, không phải của mỗi app, và bạ có thể triển khia các hành vi kahcs nhau tại mỗi điểm trên vòng đời của mỗi *Activity*

Activity khởi tạo : onCreate()

```
@Override
public void onCreate(Bundle saveInstanceState) {
        super.onCreate(savedInstanceState);
}
```

Activity của bạn bước vào trạng thai khởi tạo khi nó được chạy lần đàu tiên, khiu một *Activity* được khởi tạo lần đầu tiên, hệ thống sẽ gọi tói *onCreate()* để khở tạo *Activity*. Đơn giản, nếu app bắt đầu một *Activity* khác với một *Intent*, hệ thống sẽ ghép nối *Intent* request với *Activity* và gọi *onCreate()* tại *Activity* mối

onCreate() là bắt buộc gọi và bạn phải triển khai nó bên trong *Activity* của bạn. Trong phương thức *onCreate()* bạn thực hiện các thao tác bắt đầu cơ bản và nên chỉ làm một lần, ví dự như cài đặt UI, đăng kí các biến cục bộ, hoặc là thiết lập các nhiệm vụ chạy nhầm.

Activity bắt đầu: onState()

```
@Override
protected void onStart() {
    super.onStart();
    // The activity is about to become visible.
}
```

Sau khi *Activity* được khởi tạo với *onCreate()*, hệ thống gọi tiếp *onStart()*, và *Activity* tới trạng thái start. *onStart()* cũng được gọi nếu như bạn dừng một *Activity* và trở lại, ví dụ như bạn click nút Back hoặc Up để trwro lại màn hình trước. Trong khi *onCreate()* chỉ được gọi một lần khi mà *Activity* khởi tạo thì *onStart()* có thể được gọi nhiều lần trong suốt quãn đời của *Activity*

Khi một *Activity* trong trạng thái bắt đầu và hiển thị trên màn hình, người dùng không thể tương tác được với nhó cho tới khi *onResume()* được gọi, *Activity* đang chạy và *Activity* ở trong foreground

Thông thường bạn triển khai *onStart()* trong *Activity* của bạn như là một các để khắc phục tại *onStop()*. Ví dụ, nếu bạn sử dụng các tài nguyên phần cũng (GPS/cảm biến), khi mà *Activity* dừng lại, bạn cần phải đăng kí lại chúng tại *onStart()*o

Activity tiếp tục: onResume()

```
@Override
protected void onResume() {
    super.onResume();
    // The activity has become visible (it is now "resumed").
}
```

Activity ở tại trạng thái tiếp tục khi mà nó đã được khởi tạo, xuát hiện trên màn hình và sẵn sàng để sử dụng. Trạng thái tiếp tục thường được gọi là trạng thái chạy, bởi vì trong trạng thái này người dùng thực sự có thể tương tác với app

Lần đầu tiên *Activity* được bắt đầu gọi *onResume()* chỉ sau *onStart()*. *onResume()* có thể được gọi nhiều lần, mỗi lần app trở lại trạng thái paused

Giống như *onStart()* và *onStop()* phải triển khia như một cặp, bạn chỉ cần triển khai *onResume()* như là một các để có thể khắc phục *onPause()*.

Activity tạm dừng

```
@Override
protected void onPause() {
    super.onPause();
    // Another activity is taking focus
    // (this activity is about to be "paused").
}
```

Trạng thái tạm dừng có thể xẩy ra trong nhiều tình huống

- *Activity* chạy ngầm, nhưng chưa stop hoàn toàn. Đây là dấu hiệu đầu tiên cho thấy người dùng rời khởi hoạt ôdngj
- *Activity* chỉ hiển thị một phần trên màn hình ví một hộp thoại hoặc một *Activity* trong suống khác phủ lên nó
- Trong chế độ nhiều cửa sổ, hoạt đọng được hiển thịt rên màn hình, nhưng một số hoạt đọng kahcs lại tập trung vào người dùng

Hệ thống gọi *onPause()* khi mà *Activity* di chuyển vào trạng thái tạm dừng. Bởi vì *onPause()* là đấu hiện đầu tiên bạn biết răng người dùng có thể rời khỏi *Activity* và bạn có thể sử dụng onPause() để dừng các hoạt cảnh / video, ...

Phương thức *onPause()* nên được thực thi nhanh chóng, dừng dùng onPause() cho các hoạt động sử dụng nhiều CPU chẳng hạn như ghi dữ liệu vào cơ sở dữ liệu. AApp có thể vẫn còn hiển thị trên màn hình và có thể đi qua trạng thái tạm dừng, và bất kì độ trễ nào khi thực thi *onPause()* có thể làm chậm quá trình chuyển sang *Activity* mới. Triển khai bất kì hoạt động nặng khi app ở trạng thái stoped

Đối với API 24, *Activity* bị pause vẫn có thể hiển thị trên màn hình, trong trường hợp này bạn muốn dừng các haotj cảnh /video và bạn vẫn muốn nhìn thấy *Activity*, bạn có thể sử dụng *inMultiWindowMode()* để thử khi app của bạn chạy trong chế độ nhiều cửa sổ

Activity dừng : onStop()

```
@Override
protected void onStop() {
    super.onStop();
    // The activity is no longer visible (it is now "stopped")
}
```

Một *Activity* trong trạng thái dừng là khi nó không còn được hiển thị trên màn hình nữa. Điều này thường bởi vì người dừng đã bắt đầu một hoạt động khác hoặc là trở lại màn hình chính. Hệ thống Android giữ lại thể hiện của *Activity* trong back stack, nếu như người dùng trợ lại hoạt động này, hệ thống sẽ restart nó lại. Nếu như tài nguyên thấp, hệ thống có thể huỷ luôn trạng thái stop.

Activity huỷ : onDestroy()

```
@Override
protected void onDestroy() {
    super.onDestroy();
    // The activity is about to be destroyed.
}
```

Khi *Activity* bị huỷ, nó bị chấm dứ hoàn toàn, và thể hiện của *Activity* được lấy lại bở hệ thống. Điều này xảy ra ở nhiều trường họp

- Bạn gọi *finish()* trong *Activity* để tắt nó một cách bình thường
- Người dùng navigate trở lại trạng thái trước
- Thiết bị hiện tại đang trong tình trạng ít bộ nhớ, nơi mà hệ thống lấy lại bất kì hoạt động nào đang dừng để có thể mở rộng tài gnuyeen hệ thống
- Câu hình thiết bị bị thay đổi

Sử dụng *onDestroy()* gần như xoá hoàn toàn bởi vậy không có thành phần nào đang chạy sau khi *Activity* ị phá huỷ

Chú ý răng trong tường hợp khi mà hệ thống bị hạn bởi hoạt động host đối với *Activity* mà không gọi phương thức này, bởi vậy bạn không nên sử dụng

onDestroy() để lưu bất kì trạng thái data cần thiết nào của *Activity*. Sử dụng *onPause()* hoặc là *onStop()*

Hoạt động restart : onRestart()

```
@Override
protected void onRestart() {
    super.onRestart();
    // The activity is about to be restarted.
}
```

Trạng thái restart là một chuyển trạng thái mà xảy ra khi một *Activity* bị tạm dựng bắt đầu lại. Trong trường hợp này, *onRestart()* được gọi giữa *onStop()* và *onStart()*. Nếu như bạn có resource cầnp ahir dùng ./ bắt đầu, các hành vi đó xảy ra ontr *onStop()* hoặc là *onStart()* hay hơn là ở trong *onRestart()*

Cấu hình thay đổi và trạng thái của *Activity*

Ở section trướcn, trong *onDestroy()* bạn học được rằng *Activity* có thể bị phá huỷ khi nà người dùng navigate trở lại haowcj là xảy ra khi code thực thi *finish()* haowcj alf hệ thống muốn free tài nguyêm. Một các khác để một *Activity* có thể bị phá huỷ là khi thiết bị thay đổi cầu hình

Cấu hình thay đổi xảy ra trên thiết bị, trong runtime, và vô hiệu hoá bố cục hiện tại và các tài nguyên khác trong *Activity* của bạn. Thông thường, cấu hình thay đổi khi mà thiết bị thực hiện xoay. Khi thiết bị xoay từ ngang thành dọc hoặc ngược lại, bố cục của app cần được thay đổi. Hệ thống rẽ tạo lại *Activity* để giúp cho *Activity* thích ứng với cáu hình mới bằng cách load các nguồn tài nguyên thay thế.

Khi mà cấu hình thay đổi xảy ra, hệ thống Android tắt hoạt động của bạn và gọi lần lượt *onPause()*, *onStop()* , *onDestroy()*. Khi mà hệ thôgns khởi taọ lại activity từ ban đầu gọi *onCreate()*, *óntart()*, *onResume()*

Lưu trữ và phục hồi dữ liệu