**Ôn tập Android**

**B1. Android Development Introduction**

**Sự phát triển của điện thoại:**

* 1876 – Alexandra Graham Bell là người đầu tiên nhận được bằng sáng chế cho điện thoại điện
* 1936 – Alfred Gross – sáng chế, được cấp bằng sáng chế cho thiết bị walkie-talkie, CB radio, Telephone Pager
* 1975 – Dr. Martin Cooper – sáng chế ra chiếc điện thoại di động thương mại đầu tiên Motorola radio phone
* 2007 – 2 hệ điều hành: Android – iOS
* Nói cách khác, điện thoại thông minh = radio + computer

**Android là gì?**

* Hệ điều hành cho thiết bị di động sử dụng mã nguồn mở dựa trên nền tảng Linux
* Được phát triển bởi Open Handset Alliance và Google
* Hệ điều hành có nhiều ứng dụng hỗ trợ cho việc gọi, nhắn tin, email, quản lý danh bạ, lịch, giải trí, trải nghiệm đa phương tiện, các dịch vụ vị trí, bản đồ, tương tác xã hội, …
* Các lập trình viên bên thứ 3 có thể sử dụng Android API để mở rộng các chức năng của thiết bị
* Google cung cấp một nền tảng chợ ứng dụng để các lập trình viên có thể tự phân phối ứng dụng của họ

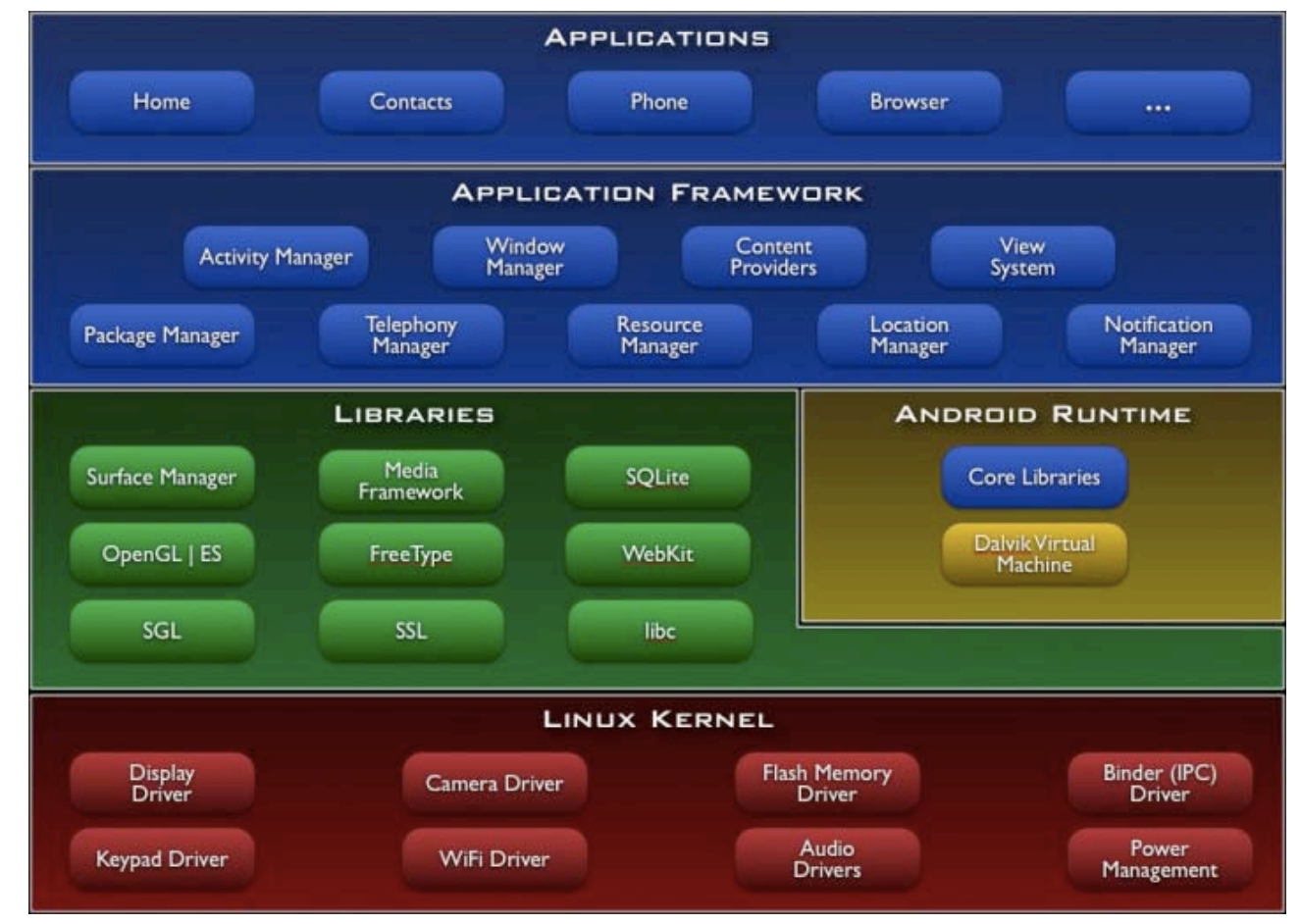
**Hiệp hội di động là gì?**

Là tập đoàn với hơn 80 doanh nghiệp và các công ty thiết bị di động

**Các thành phần (phần cứng và phần mềm) của android:**

* Dalvik virtual machine (đã được thay thế bởi ART, từ Android 4.4 trở lên)
* Integrated browser (WebKit)
* Graphic Capabilities (phụ thuộc vào tùy chỉnh phần cứng)
* SQLite cho việc lưu trữ dữ liệu
* Media support (audio / video)
* GSM Telephony (phụ thuộc phần cứng)
* Bluetooth, EDGE, 3G, 4G, NFC, Wifi (phụ thuộc các nhà sản xuất phần cứng)
* Camera, GPS, la bàn, cảm biến gia tốc, con quay hồi chuyển, cảm biến áp suất, ánh sáng, cảm biến nhịp tim, cảm biến vân tay, … (phụ thuộc vào phần cứng)
* Các công cụ phát triển phần mềm và các framework của ứng dụng (máy ảo, debugging, Android Studio, plugins cho IDE, …)

**Cấu trúc của Android:**



**B2. Android Intents**

Intent là 1 request cho các dịch vụ được cung cấp bởi Android

Intent được tạo ra bởi nhiều thành phần bao gồm:

* Các action hay service mong muốn
* Dữ liệu data
* Category của thành phần sẽ chịu trách nhiệm cho việc xử lý intent đó và cung cấp các hướng dẫn cho việc khởi chạy 1 activity mong muốn

|  |  |
| --- | --- |
| **Action** | **Data** |
| Các action để thực hiện:  ACTION\_VIEW  ACTION\_EDIT  ACTION\_MAIN | Dữ liệu để có thể xử lý, vd thông tin liên lạc trong danh bạ, được thể hiện dưới dạng Uri |

Vd:

ACTION\_VIEW content://contacts/1 – Hiển thị thông tin của người trong danh bạ có id = 1

ACTION\_DIAL content://contacts/1 – Hiển thị trình quay số với thông tin của người có id = 1

ACTION\_VIEW <tel:123> – Hiển thị trình quay số với các số cho trước

ACTION\_DIAL <tel:123> – Hiển thị trình quay số với các số cho trước

ACTION\_EDIT content://contacts/1 – Chỉnh sửa thông tin về người trong danh bạ có id = 1

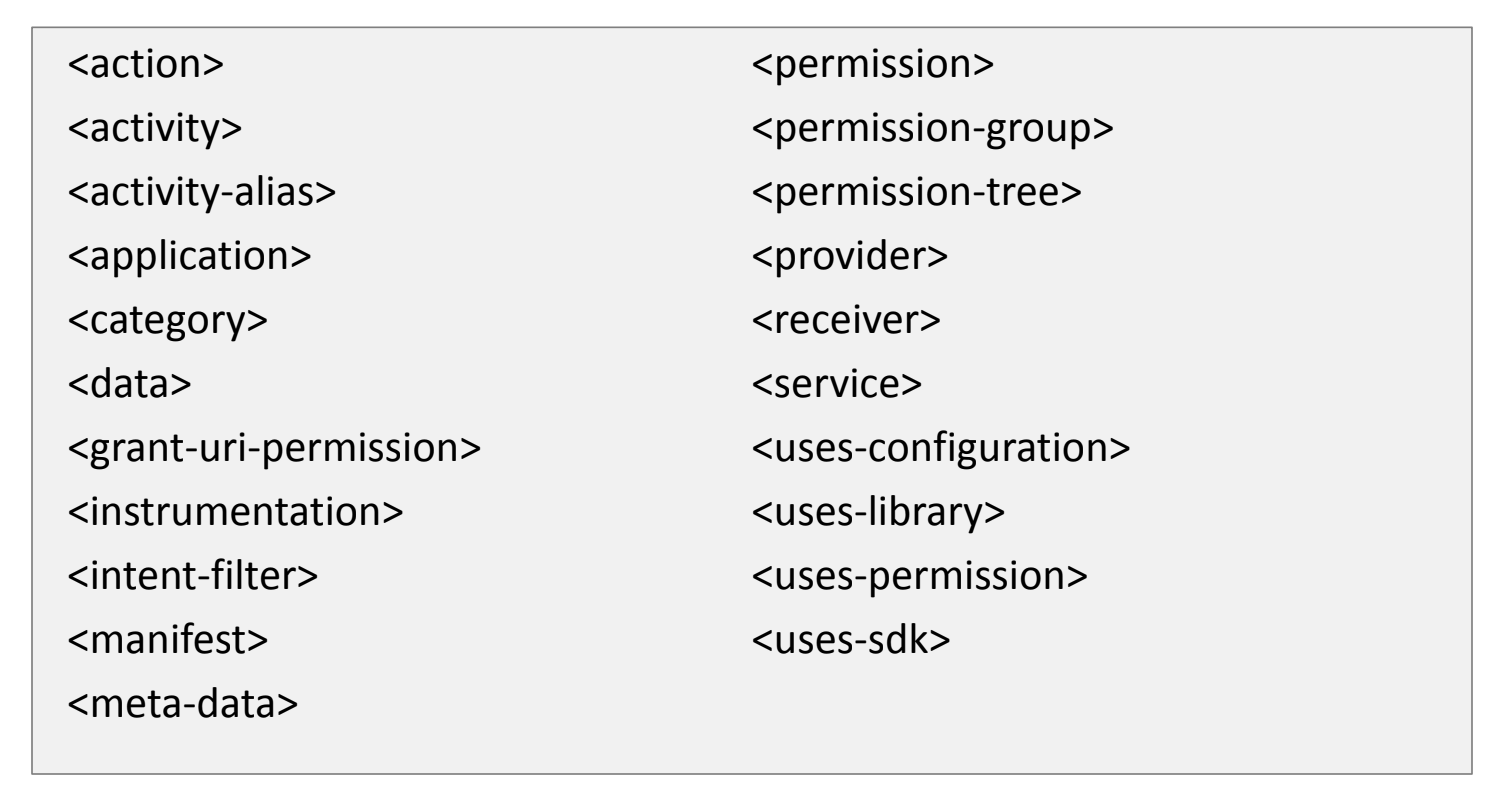
ACTION\_VIEW content://contacts/ - Hiển thị danh sách trong danh bạ

**Built-in Intent**



**Android Manifest**

Gồm các thẻ tag XML:



**Android App’s Anatomy**

* Ứng dụng Android thường được tạo ra bởi ngôn ngữ lập trình Java
* Ứng dụng phải được import các thư viện của Android (android.jar, maps.jar, …) để có thể có các chức năng cần thiết phục vụ bên trong hệ điều hành android
* Ứng dụng Android được tạo nên bởi nhiều thành phần: user-defined classes, android jars, thư viện bên thứ 3, XML file định nghĩa giao diện UI, strings, databases và cuối cùng là Manifest tóm tắt toàn bộ cốt lõi và các permissions được yêu cầu trong ứng dụng
* Các thành phần trong ứng dụng được compiler đóng gói và tạo thành 1 dạng Android Package (.apk file)
* Giống như các file .class trong Java, file .apk là những phiên bản byte-code của ứng dụng mà cuối cùng sẽ được thực thi bởi trình biên dịch bên trong Dalvik Virtual Machine (DVM) hoặc Android-Runtime Engine (ART)

**Dalvik Virtual Machine và Android Runtime**

* Dalvik Virtual Machine là 1 môi trường runtime (JIT – Just In Time) tương tự như Oracle’s Java Virtual Machine JVM, dùng để biên dịch Android byte-code khi nó cần thiết
* ART (được giới thiệu từ hệ điều hành Android 4.4) là 1 môi trường Ahead-of-Time (AOH) sẽ compile code trước khi nó được cần dùng
* ART đem lại:
* Cải thiện hiệu năng và cho thời lượng pin tốt hơn
* Cải thiện garbage collection
* Các công cụ debugging tốt hơn
* Cải thiện việc chuẩn đoán chi tiết hơn khi gặp exceptions hoặc crash

|  |  |  |
| --- | --- | --- |
| Codename | Version | API level/NDK release |
| Pie | 9 | API level 28 |
| Oreo | 8.1.0 | API level 27 |
| Oreo | 8.0.0 | API level 26 |
| Nougat | 7.1 | API level 25 |
| Nougat | 7.0 | API level 24 |
| Marshmallow | 6.0 | API level 23 |
| Lollipop | 5.1 | API level 22 |
| Lollipop | 5.0 | API level 21 |
| KitKat | 4.4 - 4.4.4 | API level 19 |
| Jelly Bean | 4.3.x | API level 18 |
| Jelly Bean | 4.2.x | API level 17 |
| Jelly Bean | 4.1.x | API level 16 |
| Ice Cream Sandwich | 4.0.3 - 4.0.4 | API level 15, NDK 8 |
| Ice Cream Sandwich | 4.0.1 - 4.0.2 | API level 14, NDK 7 |
| Honeycomb | 3.2.x | API level 13 |
| Honeycomb | 3.1 | API level 12, NDK 6 |
| Honeycomb | 3.0 | API level 11 |
| Gingerbread | 2.3.3 - 2.3.7 | API level 10 |
| Gingerbread | 2.3 - 2.3.2 | API level 9, NDK 5 |
| Froyo | 2.2.x | API level 8, NDK 4 |
| Eclair | 2.1 | API level 7, NDK 3 |
| Eclair | 2.0.1 | API level 6 |
| Eclair | 2.0 | API level 5 |
| Donut | 1.6 | API level 4, NDK 2 |
| Cupcake | 1.5 | API level 3, NDK 1 |
| (no codename) | 1.1 | API level 2 |
| (no codename) | 1.0 | API level 1 |

**Các công cụ xây dựng ứng dụng Android**

* Eclipse + ADT
* Android Studio
* NetBeans + Android

**B3. Application’s Life Cycle**

**Thành phần cốt lõi (Core components)**

* Thành phần cốt lõi là những class nguyên thuỷ hoặc các thành phần tạo nên ứng dụng
* Ứng dụng android có chứa bao gồm một hoặc nhiều các thành phần cốt lỗi. Các thành phần hoạt động hợp tác cùng nhau, mỗi thành phần đều đóng góp một cách để hoàn thiện task được thực hiện bởi ứng dụng
* Mỗi thành phần cốt lõi cung cấp một loại chúc năng và có 1 chu trình sống riêng biệt. Chu trình sống định nghĩa các thành phần được tạo ra, chuyển tiếp và bị hủy như thế nào

**Có 4 loại thành phần cốt lõi:**

* Activity
* Service
* Broadcast Receiver
* Content Provider

**Activity**

* Một object Activity gần tương đồng với WindowsForm. Nó thường được biểu diễn một giao diện người dùng GUI, nơi mà dùng để hiển thị, thu thập dữ liệu, cung cấp các chức năng hoạt động
* Một ứng dụng android thường chứa 1 hoặc nhiều Activity
* Ứng dụng phải được thiết kế sao cho có 1 Activity là main task hoặc entry point. Actiivty đó sẽ được thực thi khi app được khởi chạy
* Activity có thể chuyển các control và data cho các activity khác thông qua quá cổng giao tiếp được gọi là Intent

**Service**

* Service là một loại đặc biệt của activity mà không có giao diện người dùng. Một service có thể active mà không cần sự chú ý của người dùng đối với sự hiện diện của nó
* Service tương tự như một tiến trình thứ 2, thường chạy nền dưới background trong 1 khoảng thời gian không xác định
* Ứng dụng sẽ chỉ chạy service của nó hoặc kết nối tới các service đang active

Vd: GPS Serivce có thể chạy ngầm dưới background để xác định thông tin vị trí từ vệ tính, tháp điện thoại hay wifi. Service này có thể gửi broadcast định kỳ tọa độ vị trí đến bất kì ứng dụng nào đang lắng nghe để lấy dữ liệu vị trí đó. Ứng dụng có thể kết nối đến tiến trình GPS Service và sử dụng dữ liệu nó cung cấp

Broadcast Receiver

* Broadcast Receiver là một listener sẽ đợi đến khi nhận được thông điệp system-wide (cả hệ thống) để làm một việc gì đó. Thông điệp đó có thể là: thông báo pin yếu, wifi connection available, cảnh báo động đất, …
* Broadcast receivers không hiển thị giao diện người dùng
* Nó thường đăng ký với hệ thống bằng việc làm đóng vai trò làm bộ lọc là chính. Khi một thông điệp được gửi broadcast trùng với key của receiver được active
* Broadcast receiver có thể phản hồi bằng cách thực thi một activity cụ thể nào đó hoặc sử dụng cơ chế notification để yêu cầu sự chú ý của người dùng

**Content Provider**

* Content Provider là 1 data-centric service mà sẽ làm đồng nhất datasets đang được available cho bất kì ứng dụng nào
* Các global datasets thông thường bao gồm: danh bạ, hình ảnh, tin nhắn, email, …
* Global datasets thường được lưu trong SQLite database (developers không cần thiết phải giỏi SQL cũng làm được)
* Các class content provider thường đuưa ra 1 chuẩn cho các hàm để các ứng dụng có thể lấy, đọc, xóa, cập nhật, thêm dữ liệu

**Component’s Life cycle**

* Mỗi ứng dụng đều được chạy trong một máy ảo của nó (Virtual Machine - VM)
* Tại bất kì thời điểm nào, sẽ có một vài máy ảo chạy song song có thể được kích hoạt (cơ chế chạy song song khác với việc chuyển đổi giữa các task)
* Không giống với tiến trình trong Windows hay Unix, một ứng dụng Android không có quản lý hoàn toàn chu kì sống của chính nó
* Ngoài ra, thông thường thì resource phần cứng đôi khi không đủ và hệ điều hành có thể hủy bất kì process nào. Các tiêu chí để lựa chọn:

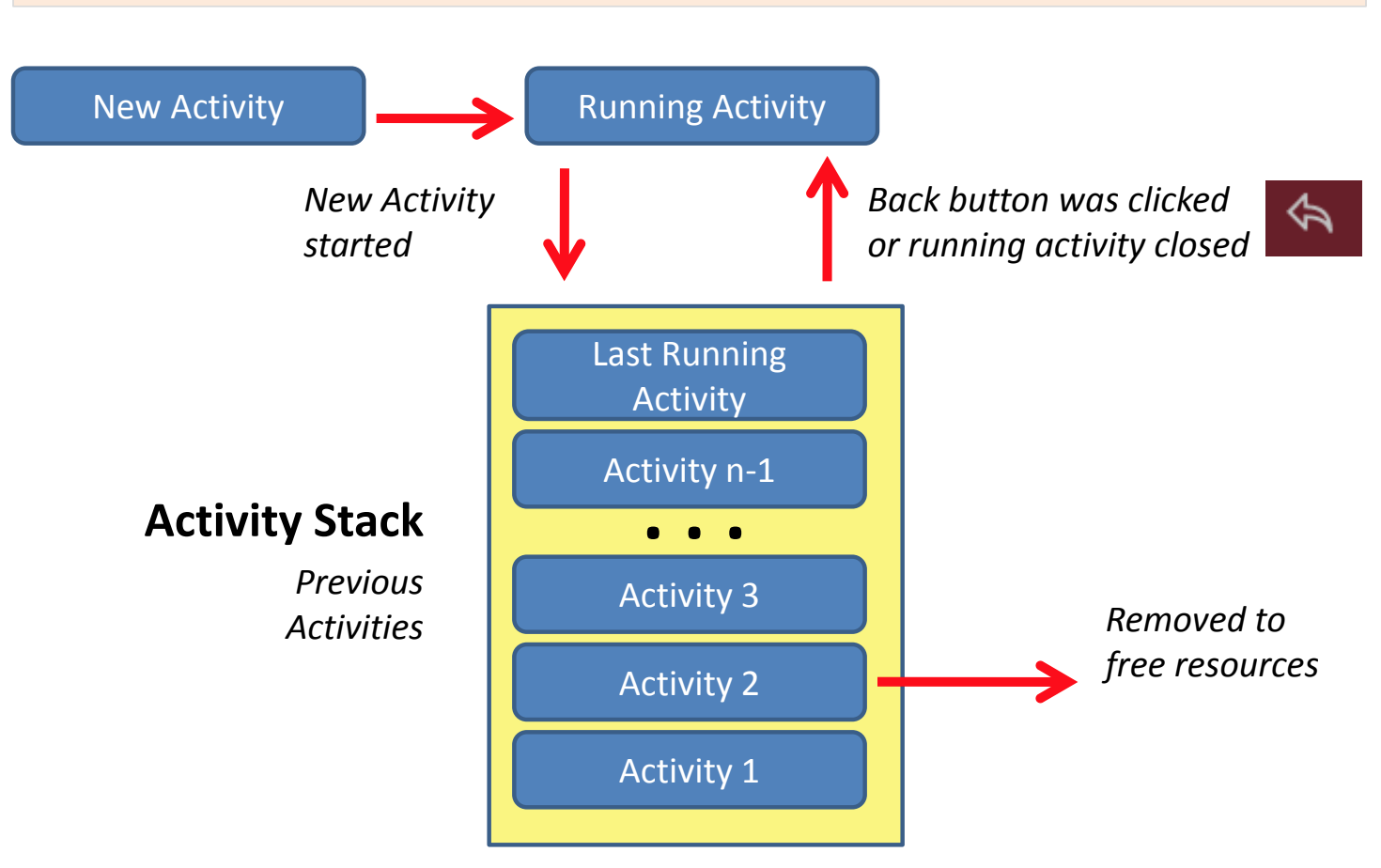
1. Số lượng và thời gian của thành phần ứng dụng đang chạy
2. Mức độ quan trọng của các thành phần đối với user
3. Lượng bộ nhớ còn trống trong hệ thống

* Các thành phần đều trải qua quá trình



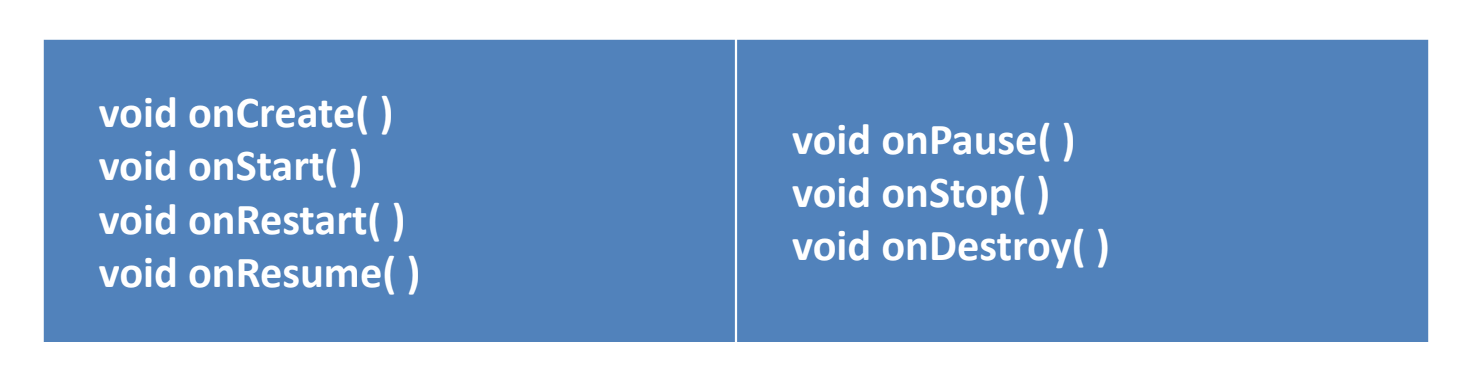
**Activity Stack**

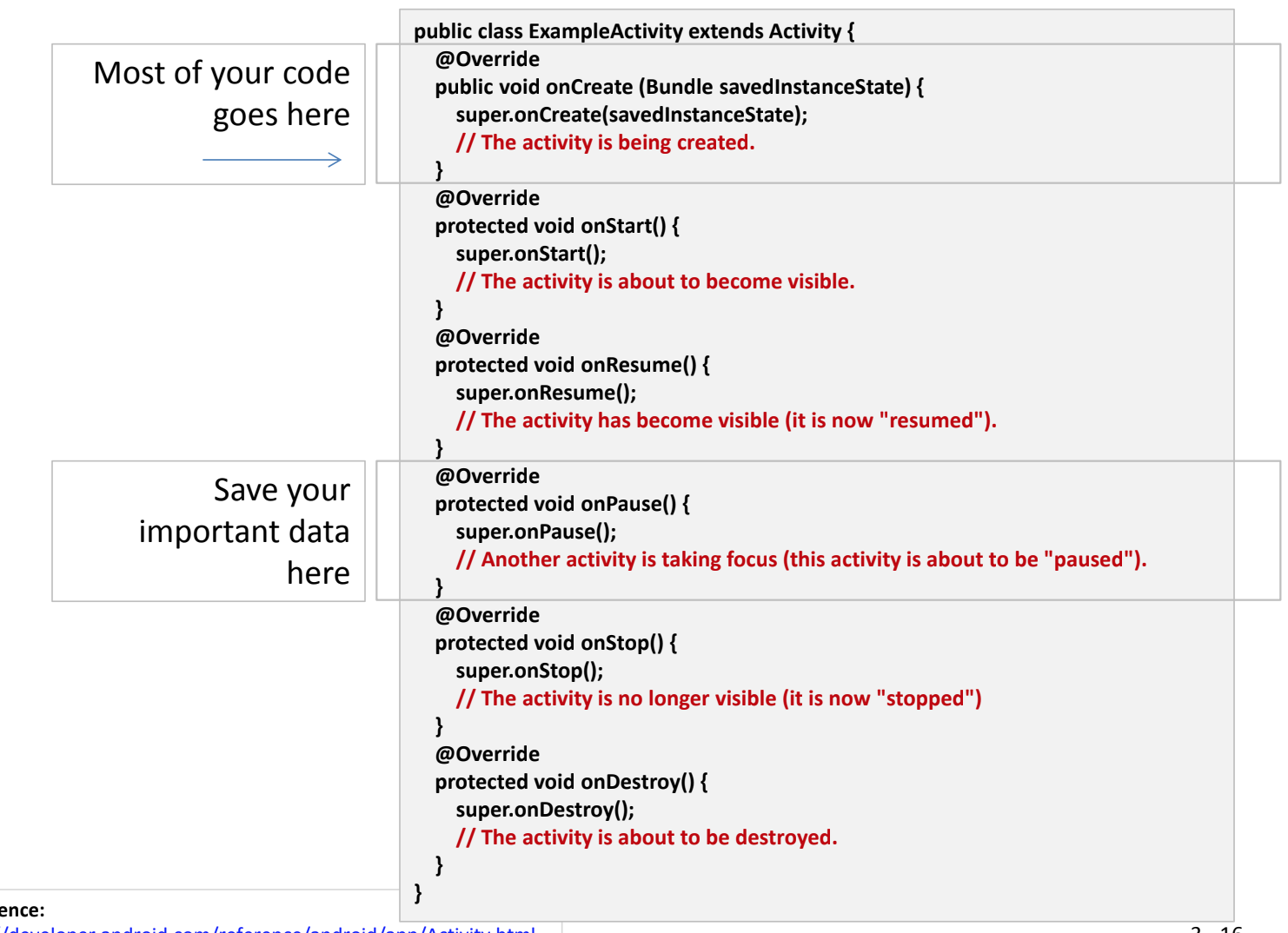
* Activities trong hệ thống được scheduled gọi là activity stack
* Khi một activity được khởi tạo, nó sẽ được đặt trên đỉnh của stack và trở thành activity đang chạy
* Các activity trước đó sẽ được đẩy xuống một level trong stack, và có thể quay lại foreground một khi activity trên nó kết thúc
* Nếu user bấm nút back, activity hiện tại sẽ bị terminated và activity trước đó trong stack sẽ dời lên và trở thành active
* Android 4.0 giới thiệu nút Recent app, để có thể chọn bất kì activity nào trong stack

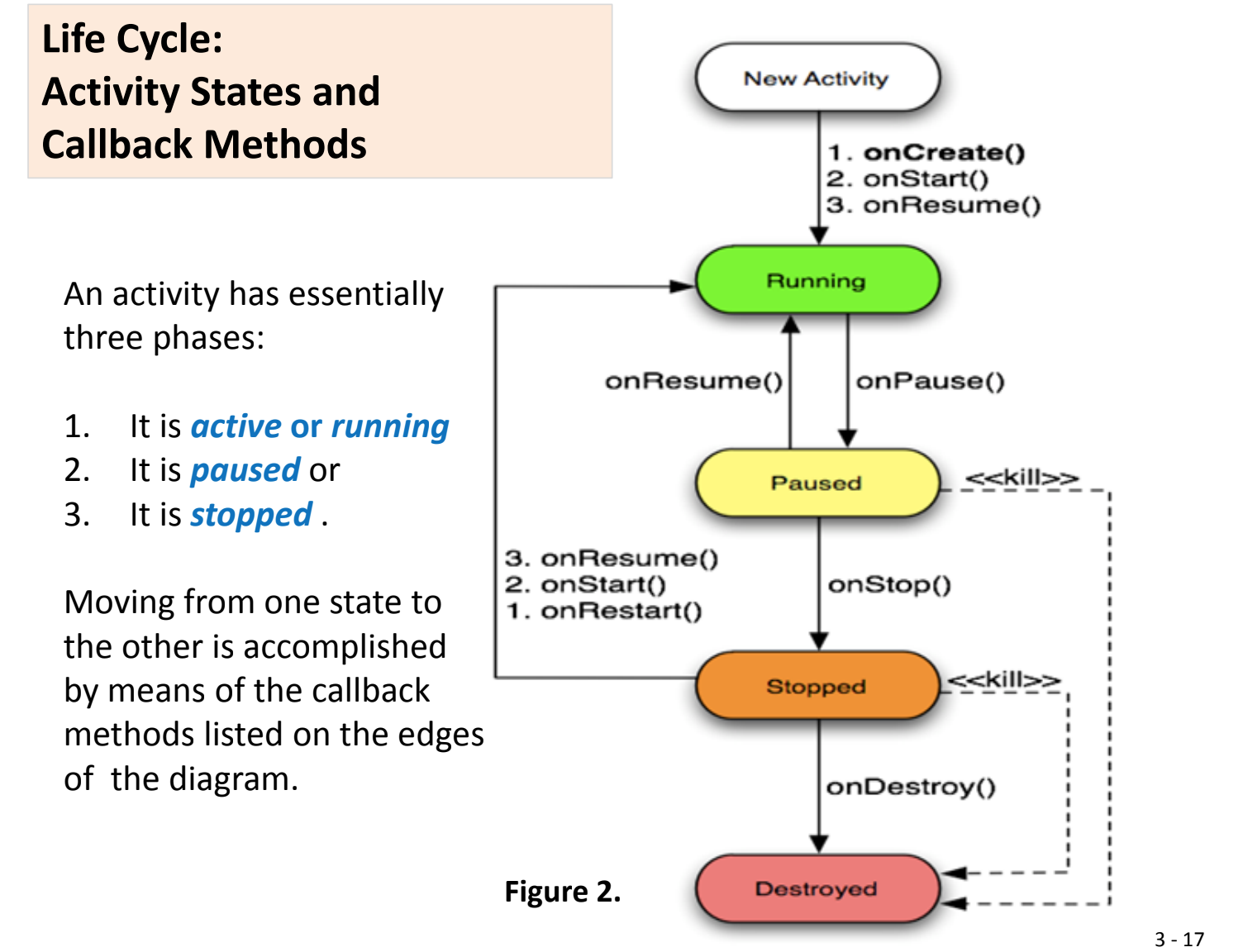


**Life cycle callbacks**

Gồm các methods:







**Các trạng thái (State)**

**\*RUNNING:**

Khi màn hình là Foreground ( Activity nằm trên cùng ứng dụng và cho phép người sử dụng tương tác)

**\*PAUSE:**

Activity bị mất focus nhưng mà vẫn nhìn thấy được Activity này (Ví dụ bạn mở một Activity mới lên dưới dạng Dialog). Trường hợp này nó vẫn có khả năng bị hệ thống tự động “XỬ” trong tình huống bộ nhớ quá ít.

**\*STOPPED:**

Activity mất focus và không nhìn thấy được (ví dụ bạn mở một Activity mới lên mà Full màn hình chẳng hạn). Trong trường hợp này nó có thể bị hệ thống “Xử” trong bất kỳ tình huống nào.

**Các hàm quan trọng trong Android Lifecycle**

* ***onCreate():*** bắt buộc – phải được implemented bởi từng activity để có thể khởi tạo. Phương thức này được thực thi một lần trong vòng đời của activity
* ***onPause():*** Đề nghị cao – nên được implement khi nào ứng dụng có dữ liệu quan trọng cần phải commit để có thể tái sử dụng

|  |  |
| --- | --- |
| ***onCreate*** | ***onPause*** |
| * Phương thức callback được gọi đầu tiên khi một activity được khởi tạo * Phần lớn mã nguồn ứng dụng được viết ở đây * Thường dùng để khởi tạo caasutrusc dữ liệu ứng dụng, liên kết các thành phần UI (button, list, …) với các Java control, listener, … * Có thể nhận dữ liệu (Bundle) chứa các thông tin, trạng thái của các activity trước đó (nếu có) * Theo sau sẽ là onStart, onResume | * Được gọi khi hệ thống chuẩn bị chuyển control đến các activity khác. Nó nên được sử dụng để ghi các dữ liệu chưa commit hay ngưng bất kì công việc nào đang chạy * Activity tiếp theo sẽ phải đợi hàm này thực thi xong mới có thể chạy * Theo sau là onResume nếu activity trở lại foreground, hoặc onStop nếu activity không còn xuất hiện (invisible) đối với người dùng * Một ứng dụng bị tạm ngưng (paused) có thể bị giết bởi hệ thống |

**Các trạng thái có thể bị giết**

* onPause, onStop, onDestroy
* onPause là trạng thái duy nhất đảm bảo được khả năng thực thi xong trước khi bị terminated
* Nên dùng onPause để ghi những dữ liệu đang chờ xử lý

**Cách để đảm bảo đồng bộ dữ liệu (Data Persistance)**

* Sử dụng SharedPreferences
* SharedPreferences là cơ chế đảm bảo đồng bộ đơn giản nhất của Android, nó được sử dụng để chứa và lấy các cặp key, value pairs. Trong đó key là 1 chuỗi và value là một trong các giá trị dạng primitive (int, float, string, …)
* Container class này sao chép cấu trúc và hành vi của HashMap trong java, tuy nhiên, khác với HashMap, nó persistent hơn
* Thích hợp cho việc chứa một lượng dữ liệu trạng thái nhỏ qua các phiên dữ liệu (session)
* SharedPreferences myPrefSettings = getSharedPreferences(MyPreferrenceFile, actMode);

**B4. Android GUI**

**MVC trong Android:**

* Model – Bao gồm Java code và các API object được dùng để đại diện các business problem và quản lý các hành vi và dữ liệu của ứng dụng
* View – Tập các màn hình mà người dùng thấy và có thể tương tác
* Controller – Được implement thông qua Android OS, chịu trách nhiệm cho việc biên dịch các input đầu vào của hệ thống và người dùng. Đầu vào có thể đến từ nhiều nguồng như trackball, bàn phím, màn hình cảm ứng, GPS chip, … và giao tiếp với Model và View (thông qua các callbacks và listeners) để có các thay đổi phù hợp

**Lớp View**

* Lớp View là thành phần đơn giản nhất mà các giao diện người dùng có thể được khởi tạo. Nó đóng vai trò như 1 container chứa các thuộc tính có thể hiển thị
* View chiếm 1 phần hình vuông trên màn hình và chịu trách nhiệm cho việc vẽ và handling events
* Widget là subclass của View. Nó được sử dụng để tạo ra các thành phần UI có thể tương tác được như button, checkbox, label,…
* Layout là các containers có cấu trúc vô hình, được sử dụng để đựng các View khác hoặc các nested layout

|  |  |  |
| --- | --- | --- |
| **LinearLayout** | **RelativeLayout** | **TableLayout** |
| Đặt các view bên trong nó theo chiều ngang hoặc chiều dọc | Là một ViewGroup cho phép người dùng có thể đặt các thành có vị trí tương đối với các thành phần khác | Là ViewGroup cho phép đặt các element bằng cách sử dụng các hàng và các cột |

**Android GUI chi tiết**

**\*FrameLayout:**

* Là dạng đơn giản nhất của GUI container
* Nó thích hợp cho việc sử dụng như container ngoài cùng để chứa các cửa sổ bên trong
* Cho phép định nghĩa kích thước của màn hình sẽ sử dụng
* Tất cả các thành phần con đều được đặt tại góc trái phía trên của màn hình

**\*LinearLayout:**

* Hỗ trợ cho việc lấp đầy các phần tử theo dạng stack theo chiều nằm ngang hoặc nằm dọc
* Nếu layout là chiều nằm dọc, các dòng mới sẽ được đặt chồng lên nhau
* Nếu layout theo chiều nằm ngang, các thành phần sẽ được đặt theo cột kế bên nhau

**\*RelativeLayout:**

* Vị trí các widget trong RelativeLayout đựa dựa vào vị trí tương đối của nó với các thành phần khác trong container cũng như là container cha nó

**\*TableLayout**

* Dùng template dạng grid để đặt các widget
* Giống như ma trận 2D, các ô trong grid được xác định bởi dòng và cột
* Cột trong tableLayout thì flexible hơn, nó có thể co lại hoặc dãn ra đề phù hợp content của nó
* Thành phần TableRơw dùng để định nghĩa 1 dòng mới mà widget có thể được đặt
* Số lượng cột trong TableRơw được xác định bởi tổng các widget được đặt cạnh nhau trên một dòng

**Android Context là gì?**

* Trong Android, Context định nghĩa một logical workspace mà ứng dụng có thể load và truy cập các resources
* Khi một widget được tạo, nó được gắn với 1 context nhất định. Bằng việc liên kết của nó với môi trường, nó có thể truy cập tới bất kì các thành viên nào nằm trên cây hierarchy
* Một ứng dụng có thể có nhiều activity. Vì thế đối với ứng dụng nhiều màn hình ta chỉ có 1 app context, và 1 context cho từng activity, có thể dễ truy cập vào những gì đang tồn tại trong cái context đó

**Based Widget:**

**\*ListViews:**

* ListView trong android là thành phần phổ biến nhất dùng để hiển thị dữ liệu được cung cấp bởi data adapter
* ListView có thể cuộn được, từng item trong base data set có thể được hiển thị như 1 dòng độc lập
* Người dùng có thể chạm vào một dòng để thực hiện lựa chọn
* 1 dòng có thể hiển thị một hoặc nhiều dòng chữ cũng như là hình ảnh

**\*GridView:**

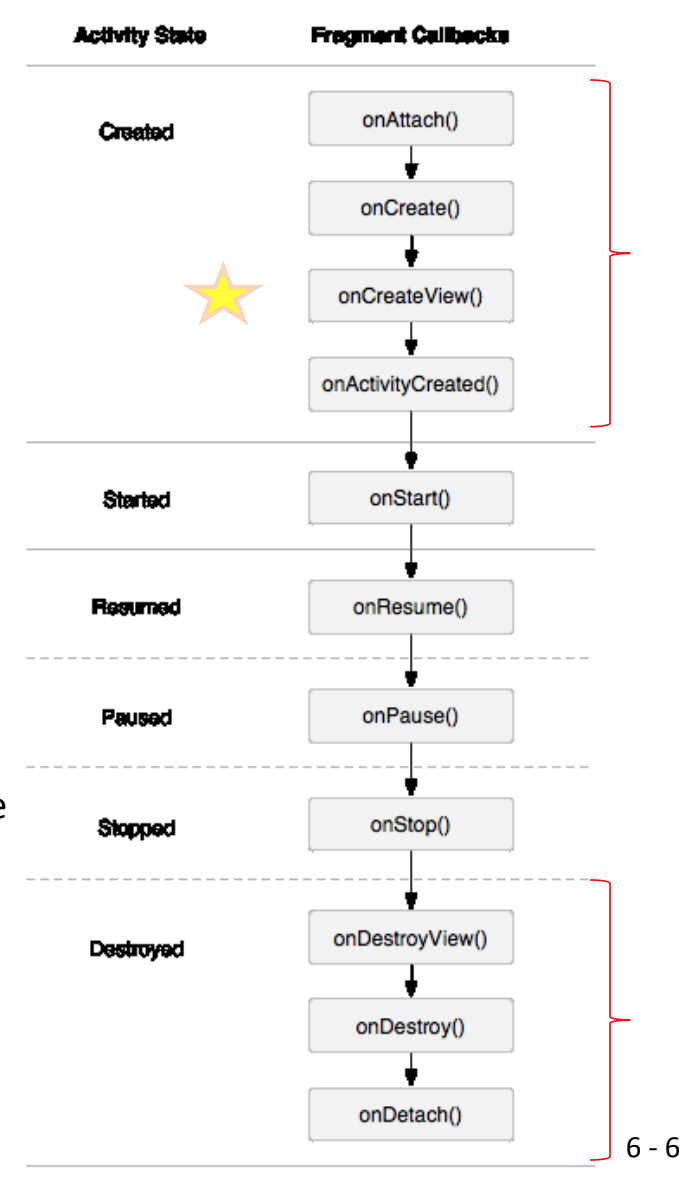
* Là một ViewGroup hiển thị danh sách các item dạng 2 chiều, có thể scroll được
* Data item hiển thị bởi grid được cung cấp thông qua data adaptẻ
* 1 ô trong grid có thể hiển thị hình hoặc chữ

**\*Fragments:**

* Fragment class cung cấp các đối tượng trược quan có thể gắn vào một phần trong giao diện ứng dụng một cách linh hoạt. Từng fragment có thể hiển thị view của chính nó và cung cấp các phương tiện để người dùng có thể tương tác với ứng dụng
* Fragment phải tồn tại trong giới hạn của một activity
* Một giao diện của host activity có thể hiển thị nhiều fragments. Trong GUI đó, từng fragment có thể visible và active
* Fragment hoạt động như một threads độc lập, nó thường xuyên hợp tác để cùng đạt một mục đích chung. Tuy nhiên, từng cái fragment có thể chạy input/output, các sự kiện và business logic của riêng nó
* Fragment có thể truy cập vào các global data được giữ trong activity mà nó thuộc về. Nó cũng có thể truyền các giá trị của nó đến activity chứa nó để có thể truyền đến các fragment khác
* Fragment cũng có life-cycle riêng
* Fragment được gới thiệu lần đầu vào android HoneyComb API 11

**Fragment Life Cycle**

* onAttach(): Xảy ra khi fragment đã kết nối với activity chủ
* onCreate(): Dùng để khởi tạo các thành phần cần thiết không trực quan của fragment
* onCreateView(): Phần lớn công việc được thực hiện ở đây. Được gọi để tạo ra vieư hierarchy đại diện cho fragment. Thường khởi tạo giao diện, định nghĩa listeners và các widget trong giao diện được khởi tạo
* onPause(): Session chuẩn bị kết thúc. Nên commit các thay đổi của dữ liệu cần thiết đề phòng trường hợp fragment sẽ khởi tạo lại
* onDetach(): Được gọi khi fragment bị ngắt khỏi activity



**Các cách fragment kết nối với activity**

**\*Dynamic Binding:**

Activity chính định nghĩa một chỗ trên GUI của nó dành cho fragment có thể gắn vào. Phần dành cho fragment đó không tồn tại mãi mãi. Activity chủ có thể thay thế fragment với một thành phần khác tại chỗ đó

**\*Static Binding:**

Trong giao diện activity khai báo 1 phần layout của nó là fragment và cung cấp rõ ràng một reference đến fragment đó bằng cụm “android:name=fragmentName”. Sự liên kết đơn giản này không đòi hỏi phải gọi hàm dựng (hoặc truyền các tham số khởi tạo). Static binding là vĩnh viễn, fragment không thể bị thay thế vào run time

**\*Multiple fragments:**

Activity chủ có thể đồng thời hiển thị nhiều fragment bằng cách kết hợp các chiến lược được diễn tả ở trên. Các fragments có thể tương tác với nhau bằng cách sử dụng activity như là trung tâm lưu trữ và chuyển tiếp

**B5. Android Intents**

* Một ứng dụng Android có thể có nhiều activity
* Manifest của ứng dụng được thiết kế để 1 activity là activity chính dùng để hiển thị cho người dùng khi ứng dụng khởi chạy (android.intent.action.MAIN)
* Thông thường, 1 activity tương ứng với một màn hình
* Một activity sử dụng setContentView() để hiển thị giao diện
* Các activity thường độc lập với nhau. Tuy nhiên nó thường hợp tác với nhau để trao đổi dữ liệu và actions
* Activities tương tác với nhau trong chế độ không đồng bộ
* Truyền control và dữ liệu từ một activity đến các activity khác thường được thực hiện bằng cách hỏi activity hiện thực thi 1 intent

Intent gần tương đương với 1 procedure trong Java

Intent gồm 2 phần chính: Action và Data

**Dùng bundle để truyền dữ liệu**

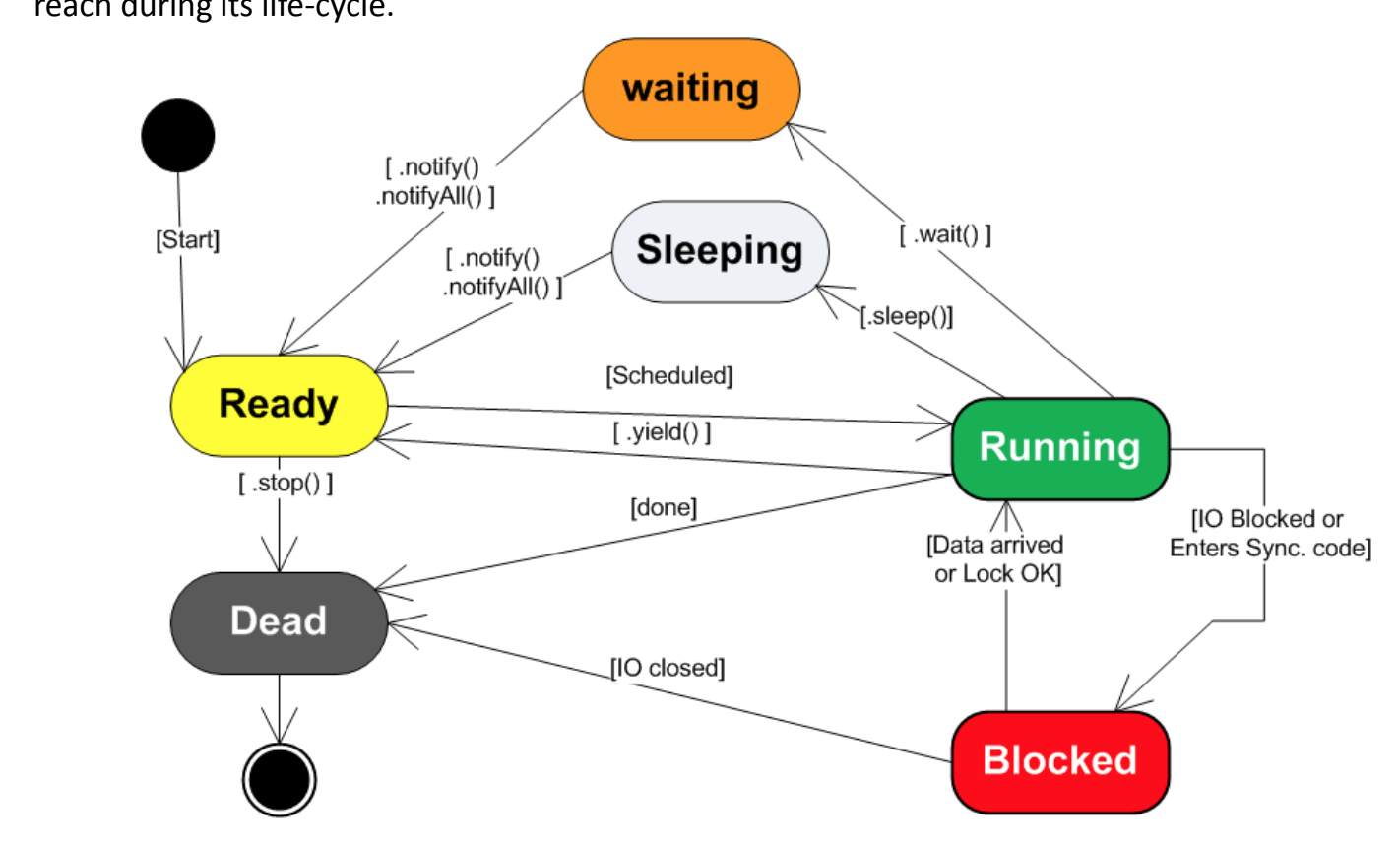
* Một Bundle là cơ hế trao đổi dữ liệu của Android sử dụng interprocess communication hiệu quả ở cả 2 phía in-process hoặc cross-process
* Một bundle về mặt khái niệm thì gần giống với HashMap trong Java. Nó cũng có một string là key ứng với một thành phần dữ liệu. Dữ liệu có thể là các data dạng primitive hoặc các object. Bundle có chức năng tương đồng với một collection <name, value>

**B6. Xử lý đồng thời**

**Android Threads**

Trong tại môt thời điểm nhất định, một ứng dụng có thể muốn làm nhiều hơn một việc tại một thời điểm. Vd: Hiển thị ảnh động, tải file lớn, và duy trì giao diện responsive khi người dùng nhập dữ liệu. Solution là cho ứng dụng chạy các hành động cùng một lúc trong các threads riêng biệt

* **JVM cung cấp kiến trúc Multi-Threading (đa luồng)** – là kết quả của việc JVM và Dalvik VM không phụ thuộc vào phần cứng
* Threads trong các máy ảo VM hoạt động và đồng bởi bởi cách sự dụng các shared objects và monitor
* Từng máy ảo có ít nhất là 1 main thread – luồng chính
* Từng thread đều có 1 stack riêng của nó. Stack được sử dụng cho việc gọi các hàm, truyền tham số và lưu trữ các biến local trong phương thức

**Lifecycle của Thread**

**Classic Java JDK Monitors (Mutex – Quản lý tiến trình bất biến)**

* Một monitor là một vùng mã nguồn quan trọng được thực hiện bởi một thread tại một thời điểm. Để implement Java Monitor, ta có thể sử dụng modifier synchronized, và có thể được khóa trên một object. Khi một thread lock một object (đọc hoặc ghi), các thread khác phải đợi cho đến khi khóa trên object đó được mở

**ReadWriteLocks**

Hiệu năng sẽ tốt hơn nếu nhiều threads cùng thực hiện đọc từ một resource dùng chung. Tuy nhiên, chỉ có thể một người được ghi. Java hỗ trợ khóa Read/Write

**Semaphores**

Counting Semaphores duy trì một pool có n threads. Nó có thể hoạt động như một người giữ cửa chỉ cho phép tối đa n thread tại một thời điểm, cũng như là cơ chế cho việc gửi các tín hiệu giữa 2 threads

**BlockingQueues**

Class BlockingQueue tạo ra một synchronized queue với bới kỳ lượng producers và consumers. Nó được implement bằng cách sử dụng các class: ArrayBlockingQueue, DelayQueue, LinkedBlockingDeque, PriorityBlockingQueue và SynchronousQueue

**Ưu điểm của Đa luồng**

* Nhiều thành phần chức năng của ứng dụng có thể được trừu tượng hóa quanh khái niệm các hành động nối tiếp hoặc song song
* Các hành động nối tiếp có thể được implement bằng cách dùng các phương thức phổ biến của hàm, trong khi các hành động song song có thể được assigned đến các threads độc lập
* Các threads có thể chia sẻ nguồn dữ liệu được giữ trong process chứa chúng
* Các ứng dụng responsive có thể dễ dàng được tạo bởi cách đặt logic control sự tương tác của người dùng lên main thread, trong khi các tiến trình chậm thì được đặt ở dưới background thread
* Một chương trình đa luồng thì sẽ chạy nhanh hơn trên một hệ thống máy tính có nhiều CPU. Hiện nay phần lớn thiết bị android đều có CPU đa nhân

**Bất lợi của đa luồng**

* Code sẽ phức tạp hơn
* Cần phải xác định, tránh, giải quyết các vấn đề deadlocks

**Handler class**

* Main thread có thể sử dụng MessageQueue của nó để quản lý các tương tác giữa main và các background threads mà nó tạo
* Message queue hoạt động như một hàng đợi ưu tiên được bảo vệ bởi semaphore với khả năng enqueue tokens chứa các messages hoặc runnables được gửi bởi các 2nd threads
* Theo thủ tục, các threads con phải request token rỗng từ queue cha mẹ của nó, lấp đầy và sau đó gửi lại cho queue cha mẹ của nó
* Trong kiến trúc Android, từng thread có một MessageQueue. Để sử dụng, một object Handler phải được tạo ra
* Handler sẽ enqueue messages và runnables cho hàng đợi message cha mẹ nó. Những request này sẽ được thực hiện theo thứ tự cũng như cách mà nó sẽ được remove khỏi message queue

**Handler hỗ trợ 2 hành động quan trọng nhất:**

1. Schedule messages và runnables để thực hiện tại một thời điểm nào đó trong tương lai
2. Để enqueue các hành động sẽ được thực hiện trên thread khác

***Chú ý: Thread không thể tác động lên UI của app trừ main thread***

====

**B7. XML và JSON**

***Enscript:*** mã hóa dữ liệu

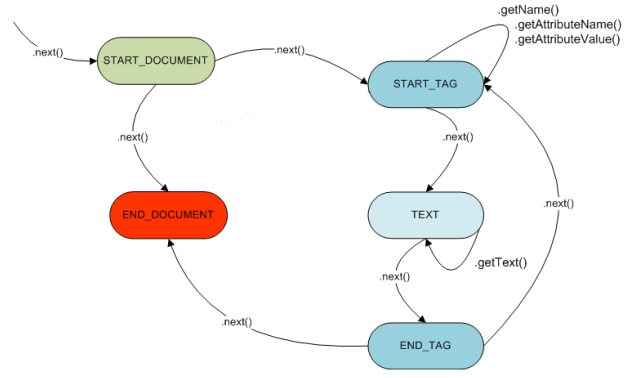
***Encode:*** chuyển đổi sang định dạng dữ liệu có thể xử lý dễ dàng

* Có 2 vấn đề cần quan tâm trong việc trao đổi dữ liệu trên internet: size và transpanrency (khả năng đọc 1 cách dễ dàng mà không cần chuyển đổi)
* XML và JSON là dạng dữ liệu text có cấu mà có thể đọc 1 cách dễ dàng.
* **XML – Extensible Markup Language:** là một tập hợp các quy tắc được đề nghị bởi W3C theo một khuôn khổ thống nhất trên toàn thế giới. Mô tả nhiều loại dữ liệu khác nhau theo định dạng mà chúng ta có thể dễ dàng đọc hiểu được.
* XML tương tự HTML và chỉ có 1 thẻ gốc và các thẻ con. Các <tags> được định nghĩa bởi người dùng.
* Có nhiều công nghệ sủ dụng XML: RSS, SOAP, Atom, XHTML, KML, Xquery, Xpath, OpenDocument, OpenMath…
* XML thường sử dụng file .xsd để mô tả các tài liệu .xml
* **Có 2 cách phân tích tài liệu XML:**
* **SAX (Simple API for XML):**

Kiểu đối tượng XmlPullParser dùng để duyệt tài liệu từ đầu đến cuối đi qua các thẻ. Dùng phương thức .next() để đi qua các thẻ, ứng với mỗi thẻ có các sự kiện tương ứng:

* START\_TAG: đã đi đến đầu thẻ
* END\_TAG: đã đi đến cuối thẻ
* TEXT: thẻ nội dung text
* END\_DOCUMENT: đã đi đến cuối tài liệu

Với phương thức getName() để lấy tên thẻ, getText() lấy dữ liệu text sau sự kiện TEXT. Ngoài ra, còn các phương thức: getAttributeCount(), getAttributeName(), getAttributeValue() để truy vấn thông tin các thuộc tính.



* **Ưu và khuyết điểm XML:** trong trường hợp cần duyệt đi qua hết tất cả các thẻ thì XML là lựa chọn hợp lý. Còn khi chúng ta chỉ quan tâm 1 vài thẻ thì XML sẽ chậm, tốn nhiều thời gian, đặc biệt với các file XML quá lớn.

**W3C-DOM:** tương tự SAX nhưng duyệt có trọng tâm. Sử dụng đối tượng DocumentBuilder để parse dữ liệu từ file xml và duyệt theo dạng cây. , mỗi thẻ là 1 node.

VD:

*NodeList[] elementList = document.getElementsByTagName(…);*

*Node node = elementList.item(…);  
node.getName()  
node.getValue()  
node.getFirstChild()  
node.getAttributes()*

* Các file xml để trong thư mục res/xml

**JSON - Javascript Object Notation**, đây là một dạng dữ liệu tuân theo một quy luật nhất định mà hầu hết các ngôn ngữ lập trình hiện nay đều có thể đọc được. Mỗi thông tin dữ liệu sẽ có 2 phần đó là **key và value** được phân cách nhau bởi dấu “,” (comas). Mỗi object nằm trong cặp dấu ngoặc nhọn {} (curly braces), mảng được nằm trong cặp dấu ngoặc vuông [] (square brackets)

**B8. RSS**

* RSS là dạng format mà được quy định trên toàn cầu để truyền thông tin cho nhau trên internet, cung cấp nội dung thông tin theo dạng tệp XML.
* Các loại tài nguyên RSS thường thấy:  
  • Tin tức • Xã hội

• Thời tiết • Marketing & quảng cáo  
• Tài chính • Blogs  
• Du lịch • Video

* Dịch vụ
* Phiên bản đầu tiên RSS được tạo ra bởi Netscape vào khoảng năm 1999.
* ***Viết tắt là “Really Simple Syndication”***Trong mỗi file RSS, thường có 1 thẻ gốc <channel> và bên trong chứa hàng loạt <item> mỗi item là một mẫu chuyện.

|  |  |  |  |
| --- | --- | --- | --- |
| **Elements** | **Description** | **Type** | **# allowed** |
| **lastMod** | Thời gian điều chỉnh mới nhất | ISO 8601:1988 Date | 0 or 1 |
| **title** | Tiêu đề | String | 0 or 1 |
| **abstract** | Mô tả ngắn tóm tắt bài viết  (Khuyến nghị 200 ký tự trở xuống) | String | 0 or 1 |
| **author** | Tác giả | String | Any |
| **publisher** | Nhà xuất bản | String | Any |
| **copyright** | Copyright | String | 0 or 1 |
| **publicationDate** | Ngày xuất bản | String | 0 or 1 |
| **logo** | Logo cho channel | Logo element | Any |
| **keywords** | Từ khóa được phân cách bằng dấu phẩy | String | Any |
| **category** | Một danh mục mà trang web này thuộc về (dưới dạng URI). | Category element | Any |
| **ratings** | Xếp hạng kênh theo một hoặc xếp hạng của nhiều dịch vụ | String | Any |
| **schedule** | Lịch trình cập nhật | Schedule element | 0 or 1 |
| **userSchedule** | Lịch trình cập nhật chỉ định người dùng | UserSchedule element | 0 or 1 |

* Các thẻ con trong thẻ <channel>

Các thẻ con trong thẻ Các thẻ con trong thẻ

<item>

* Các thẻ con trong thẻ <item>

|  |  |
| --- | --- |
| **Element** | **Description** |
| **title** | Title của mẫu chuyện |
| **link** | URL của mẫu chuyện |
| **description** | Tóm tắt (Nếu có nhu cầu sử dụng các thẻ html như <b/> <i/> thì để trong 1 thẻ <![CDATA[…]]> VD: <description><![CDATA[ This is <b>bold</b> ]]></description> |
| **author** | Địa chỉ email của tác giả |
| **category** | Một danh mục mà trang web này thuộc về |
| **comments** | URL của một trang cho bình luận, nhận xét ​​liên quan đến các mục. |
| **enclosure** | Mô tả một đối tượng media được gắn vào mục. |
| **guid** | Một chuỗi xác định duy nhất mục này. |
| **pubDate** | Cho biết ngày được xuất bản. |
| **source** | Nguồn thông tin tài nguyên |

* Sử dụng đối tượng DocumentBuilder để parse dữ liệu và duyệt theo dạng cây, mỗi thẻ là 1 node.
* Để lấy dữ liệu từ trên mạng, sử dụng package của java là java.net hoặc thư viện apache HttpClient.
* Cách dùng lớp HttpUrlConnection trong package java.net:

1. Khởi tạo kết nối thông qua lớp HttpURLConnection
2. Thiết lập request (URI bao gồm header, credentials, content, cookies…)
3. Đọc phản hồi (thông qua hàm getInputStream ())
4. Ngắt kết nối sau khi đọc được phản hồi

* Cần khai báo <uses-permission android:name="android.permission.INTERNET" /> trong file android manifest khi có sử dụng internet.

**B9. Services**

* Services là một trong 4 thành phần chính trong android:
* Chạy ngầm
* Không tương tác với người dùng
* Chạy với thời gian bất định (không dừng)
* Chạy trong main thread
* Chủ yếu dùng để thực hiện các tác vụ nặng liên quan nhiều CPU (mp3 playback) hoặc blocking (netwoking)
* Có thể phát sinh các tiêu trình con để xử lý tác vụ (tương tự activity)
* Khi tạo service cần khai báo trong android mainifest (tương tự activity)
* Khi service thực hiện xong 1 chu ký tính toán, sẽ phát 1 thông điệp kèm trong Intent (tín hiệu)

**Có 2 cách để khởi chạy service:**

* startService(): có thể start 1 service nhiều lần dù đã đang chạy nhưng chỉ có thể gọi hàm stopService() 1 lần.
* bindService(): kết nối đến một service khác đang chạy.

**Chu kỳ sống:** chỉ có thể tiến không thể quay lui như activity, gồm các giai đoạn (phương thức): onCreate 🡢 onStart 🡢 onDestroy.

* onCreate(): setup cấu hình cài đặt…
* onStart(): khởi chạy service
* onDestroy(): nhả ra tất cả các tài nguyên

**Broadcast Receiver:**

* Là 1 trong 4 thành phần quan trọng trong android.
* Là 1 lớp có nhiệm vụ lắng nghe các intent (tính hiệu toàn cục trong android) từ service của app mình hay bất kì app nào khác.

Cần thực hiện đăng ký để có broadcast receiver, có 2 cách đăng ký:

* Đăng ký động: gọi phương thức registerReceiver()
* Đăng ký tĩnh: khai báo thông qua thẻ <receiver> trong android manifest và nằm trong đó sẽ kèm 1 thẻ <intent-filter> khai báo cho biết tính hiệu mà broadcast receiver này có thể xử lý

**Chu kỳ sống:**

* Hoạt động: khi phương thức onReceive được gọi
* Bất động: khi phương thức onReceive thực hiện xong các lệnh (Không bị hủy, vẫn luôn tồn tại trừ khi hủy trên code)

*void onReceive (Context mContext, Intent broadcastMsg)*

**Có 2 loại broadcast receiver:**

* Normal broadcasts: lả các broadcast receiver tại 1 thời điểm có thể cùng nhận tín hiệu từ 1 service và xử lý song song.
* Ordered broadcasts: là broadcast sau khi xử lý, lan truyền kết quả đến 1 broadcast khác để xử lý tiếp và cứ thế tiếp tục đến khi hoàn tất hoặc khi hàm abortBroadcast() được gọi sẽ dừng. Tại 1 thời điểm chỉ có 1 broadcast receiver thực hiện xử lý công việc và có thể ưu tiên broadcast receiver nào xử lý trước thông qua thuộc tính *android:priority*

**B10. SQL Databases**

**SQLite:** **DBMS – Database Management System**

**Đặc điểm:**

* Hỗ trợ transaction
* Chương trình nhỏ (<400 Bytes)
* Typeless (Không có kiểu DL chỉ có kiểu String)
* Serverless (Dùng phía Client)
* Không cần cài đặt cấu hình
* Mã nguồn mở
* Kế thừa chuẩn SQL-92
* Hỗ trợ 1 phần trigger và 1 số kiểu truy vấn phức tạp (Ngoại trừ: right/ full outer joins, grant/revoke, updatable views)
* Không hỗ trợ ràng buộc khóa ngoại
* Không có các kiểu DL chỉ có kiểu String
* Không ràng buộc gì cả, tự quản lý dữ liệu của mình.

**Cách tạo CSDL SQLite**

*SQLiteDatabase.openDatabase(myDBPath, null, SQLiteDatabase.CREATE\_IF\_NECESSARY);*

CREATE\_IF\_NECESSARY, nếu chưa có database thì sẽ tự tạo database, ngoài ra còn có các cách mở database theo các flag sau: OPEN\_READWRITE, OPEN\_READONLY.

**Các tham số:**

* Path: đường dẫn đến nơi để tạo hoặc chứa file database
* Factory: dùng để khởi tạo con trỏ khi câu truy vấn được gọi, có thể để null
* Flags: Cờ để chọn phương thức truy cập database
* Returns: trả về database
* Throws: SQLiteException nếu không mở được database

**Đường dẫn:**

* Sử dụng thẻ nhớ bên ngoài (/storage/sdcard/…):

*String storagePath = Enviroment.getExternalStorageDirectory().getPath();*

* Sử dụng file bộ nhớ trong hệ thống ứng dụng (data/data/<Tên package>):

File storagePath = getApplication().getFilesDir();

String myDbPath = storagePath + “/” + “<Tên thư mục chứa database>;

**Khi tạo database hoặc muốn sử dụng database trong SDCard thì phải xin cấp quyền trong file Manifest:**

<uses-permission android:name=”android.permission.WRITE\_EXTERNAL\_STORAGE” />

<user-permission android:name=”android.permission.READ\_EXTERNAL\_STORAGE” />

**Lưu ý:**

* Database được tạo trong hệ thống ứng dụng data/data/<tên package> sẽ không truy cập được bởi ứng dụng khác.
* Lưu database trong SDCard thì sẽ được public.

**Phương thức mở và tạo database khác:**

*SQLiteDatabase db = this.openOrCreateDatabase(“<Tên DB>”, MODE\_PRIVATE, null);*

* Dựa vào lệnh trên, ứng dụng sẽ mở hoặc tạo database theo đường dẫn sau:

*/data/data/<Tên package>/<Tên DB>*

* Tất cả các thành phần trong ứng dụng đều có thể truy cập db này.
* Một số giá trị MODE khác: MODE\_WORD\_READABLE, MODE\_WORD\_WRITEABLE (Không dùng được trong API 17)

**Các loại câu lệnh SQL:**

* Có 2 loại query là: Action queries và Retrieval queries
* Retrieval queries: thường để thực hiện lệnh select truy vấn trong sql
* Action queries: thường để thực các lệnh thêm, xóa, sửa trong sql
* Các câu lệnh action queries nên để trong transaction để đảm bảo tính nhất quán trong dữ liệu.

*db.beginTransaction();*

*Try {  
 db.setTransactionSuccessfull(); // commit*

*} catch (SQLiteException e) {*

*// report problem*

*} finally {*

*db.transactionEnd();*

*}*

* Dùng phương thức execSQL(…) để thực thi các câu lệnh SQL action queries
* Tất cả các câu lệnh execSQL đểu phải bỏ trong câu lệnh try…cactch…finally…
* Trong SQLite, tất cả các bảng đều có cột RowID bắt đầu từ 0, được tạo sẵn và có thể gọi để sử dụng.

**Retrieval queries** là các câu truy vấn dạng select. Có 2 loại:

* **Raw queries:** là các câu lệnh sql tự viết. Ưu điểm: xử lý, join được trên nhiều bảng. Khuyết điểm: khó, phức tạp, đòi hỏi nắm vững các câu lệnh sql. (Cú pháp: db.rawQuery(…))
* **Simple queries:** cung cấp sẵn các phương thức nhưng chỉ có thể tương tác trên 1 bảng. (Cú pháp: db.query(…))
* Trong SQLite, sử dụng cursor con trỏ để duyệt các dòng trong bảng dữ liệu kết quả trả về. Có thể xử lý tuần tự hoặc random, tốc độ chậm, chỉ xử lý trên 1 dòng dữ liệu trong 1 thời điểm và chỉ có thể read only chỉ để đọc dữ liệu, không thể thay đổi nội dung.
* Giá trị các kiểu dữ liệu của cột:

0: NULL, 1: INT, 2: FLOAT, 3: STR, 4: BLOD, 5: UNK

**Action queries:** thường để thực các lệnh thêm, xóa, sửa trong sql (insert, update, delete, create table, drop table)

***Cú pháp:*** db.execSQL(…) không có sử dụng được lệnh db.rawQuery(…) hoặc sử dụng các hàm có sẵn:

* public long **insert**(String table, String nullColumnHack, ContentValues values ) trả về RowID của dòng vừa insert
* public int **update**(String table, ContentValues values, String whereClause, String[] whereArgs) trả về số dòng bị ảnh hưởng  
  public int **delete**(String table, String whereClause, String[] whereArgs) trả về số dòng bị ảnh hưởng

VD:  
*ContentValues myArgs= new ContentValues();  
myArgs.put("name", "ABC");  
myArgs.put("phone", "555-7777");*

*long rowPosition = db.insert("tblAMIGO", null, myArgs);*

null: bắt buộc nhập dữ liệu cho tất cả các cột. Còn cột nào không muốn nhập dữ liệu thì đổi null thành tên của cột đó.