**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

🙤🙧🟍🙥🙦



**PHÁT TRIỂN ỨNG DỤNG TRÊN THIẾT BỊ DI ĐỘNG**

**NT118.J12**

**BÀI TẬP NHÓM TUẦN 1**

* **GV: Thái Huy Tân**

|  |  |  |
| --- | --- | --- |
| Nhóm 11 | | |
| **STT** | **Họ tên** | **MSSV** |
| 1 | Trương Thị Bích Chi | 15520062 |
| 2 | Lê Hồng Ngọc | 15520553 |
| 3 | Đỗ Trọng Nhân | 15520566 |

**TP. HỒ CHÍ MINH – 20/9/2018**

# so sánh ART và Dalvik virtual machine?

**ART và Dalvik đây chính là tên của 2 loại máy ảo thiết yếu cho sự hoạt động của hệ điều hành Android. Nói cách khác, ART và Dalvik là 2 trình biên dịch phổ biến dùng để viết chương trình trên các thiết bị di động Android. Tuy nhiên, một thiết bị Android không thể chạy cùng lúc cả 2 trình biên dịch này.**

Dalvik là phần mềm chịu trách nhiệm thực thi gần như tất cả các ứng dụng Android, Google sẽ phải tìm cách giúp máy ảo Android của mình có thể bắt kịp với các tiến bộ của ngành công nghiệp phần cứng. Dần dần, Google đã trang bị cho Dalvik một bộ biên dịch JIT (Just-in-time: biên dịch mã nguồn ngay trước khi thực thi), khả năng chạy nhiều thread (chạy song song nhiều luồng) cùng vô số thay đổi nhỏ khác.

ART được thiết kế để tương thích hoàn toàn với mô hình biên dịch bytecode của Dalvik. Do đó, khi nhìn từ góc độ nhà phát triển ứng dụng, chúng ta sẽ không cần phải viết mã nguồn riêng cho Dalvik rồi viết lại cho ART. Sự thay đổi lớn nhất về thiết kế mà ART mang tới là khả năng biên dịch code theo mô hình AOT (Ahead of Time) thay vì JIT (Just in Time) như trước đây.

Điều này có ý nghĩa là bộ runtime mới của Android không còn cần phải dịch từ bytecode sang mã máy mỗi lần chạy ứng dụng. Thay vào đó, ART sẽ chỉ biên dịch bytecode sang mã máy đúng một lần duy nhất. Quá trình thực thi ứng dụng trong các lần chạy tương lai sẽ được thực hiện từ các đoạn mã máy đã được tạo ra từ trước.

* Lợi ích của ART:
* Ngắn gọn: các đoạn code sẽ có thể lược bỏ các đoạn mã có thể bị coi là thừa nhưng vẫn bắt buộc phải có nếu sử dụng kiến trúc cũ, ví dụ như check exception (kiểm tra và bắt trước các lỗi có thể xảy ra trong quá trình chạy).
* Hiệu quả: Quá trình gọi hàm hoặc gọi interface cũng sẽ nhanh hơn đáng kể. Quá trình này sẽ được thực hiện bên trong phần mềm “dex2oat” – phiên bản thay thế cho dexopt trên Dalvik. Các file odex (file dex đã được tối ưu) cũng sẽ bị thay thế bằng file ELF.
* Tiết kiệm bộ nhớ: Do ART biên dịch các file thực thi ELF, nhân của Android giờ có thể quản lý các trang của code page (tập mã hóa ký tự của 1 ngôn ngữ) một cách hiệu quả hơn. Nhờ đó mà hệ điều hành di động của Google có thể quản lý bộ nhớ tốt hơn và tốn ít RAM hơn.
* Tiết kiệm pin: Ảnh hưởng của ART tới thời lượng pin có lẽ cũng là rất đáng kể, do hệ điều hành không cần phải biên dịch mã nguồn trong quá trình chạy ứng dụng nữa. Nhờ đó, quá trình chạy ứng dụng sẽ giảm được đáng kể sức ép lên CPU, giúp tiết kiệm pin tốt hơn.

- Hạn chế của ART: Điểm yếu lớn nhất của ART so với ELF là quá trình biên dịch mã sẽ tốn nhiều thời gian hơn. Do đó, quá trình khởi động điện thoại cũng như lần bật ứng dụng đầu tiên sẽ tốn nhiều thời gian hơn so với ứng dụng tương đương trên Dalvik. Người dùng thông thường sẽ chỉ phải tốn thêm thời gian chờ đợi trong lần đầu cài đặt. Trải nghiệm sử dụng sau đó sẽ trở nên mượt mà hơn rất nhiều.

# Tìm hiểu và giải thích typical flow của Android:

***1) Applications là gì?***

– Bạn hiểu nôm na như sau: Mỗi một Android Project khi bạn biên dịch thành công thì sẽ được đóng gói thành tập tin **.apk**, tập tin**.apk**được gọi là một Application (Ứng dụng cụ thể nào đó – Ví dụ như ứng dụng chống tin nhắn rác, ứng dụng tìm đường đi ngắn nhất, ứng dụng đăng ký học phần bằng sms …. )

***2) Activities là gì?***

– Thông thường trong một ứng dụng (Application) sẽ có một hoặc nhiều Activity

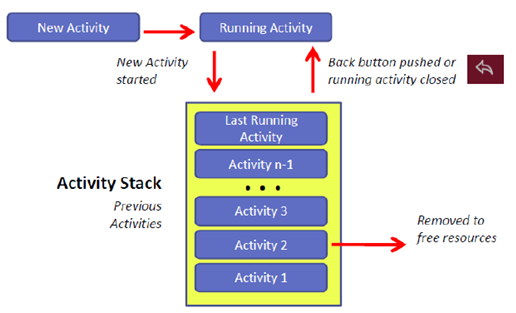
– Mỗi một Activity này sẽ có một vòng đời riêng độc lập hoàn toàn với các Activity khác.

– Mỗi một Activity muốn được triệu gọi trong ứng dụng thì bắt buộc nó phải được khai báo trong Manifest.

***3) Activity Stack là gì?***

– Tương tự như các ngôn ngữ lập trình khác, Activity Stack hoạt động theo cơ chế LIFO (LAST IN FIRST OUT)

-Mỗi một Activity mới được mở lên nó sẽ ở bên trên Activity cũ, để trở về Activity thì chỉ cần nhấn nút “Back” để trở về hoặc viết lệnh. Tuy nhiên nếu nhấn nút Home rồi thì sẽ không thể dùng nút “Back” để quay lại màn hình cũ được.



– Ở đây bạn chú ý là có 2 kiểu mở Activity mới :

a) Mở Activity mới lên làm che khuất toàn bộ Activity cũ (không nhìn thấy Activity cũ): sảy ra sự kiện **onPause** rồi **onStop** đối với Activity cũ

b) Mở Activity mới lên làm che khuất một phần Activity cũ (vẫn nhìn thấy Activity cũ): Sảy ra sự kiện **onPause**với Activity cũ.

– Khi quay trở về Activity cũ thì sau khi thực hiện xong các hàm cần thiết, chắc chắn nó phải gọi hàm **onResume** để phục hồi lại trạng thái ứng dụng

– Như vậy ta thường lưu lại trạng thái của ứng dụng trong sự kiện **onPause** và đọc lại trạng thái ứng dụng trong sự kiện **onResume.**

***4) Tasks là gì?***

– Hiểu đại khái Task là khả năng  thực hiện một công việc nào đó giữa các Ứng dụng với nhau, cụ thể là các Activity

–Ví dụ: ứng dụng tin nhắn điện thoại, cho phép mở danh bạ để tìm kiếm người dùng nhận tin nhắn. 2 ứng dụng không liên quan nhau. Sau khi lấy được người dùng nhận tin nhắn, ứng dụng danh bạn sẽ ngừng hoạt động.

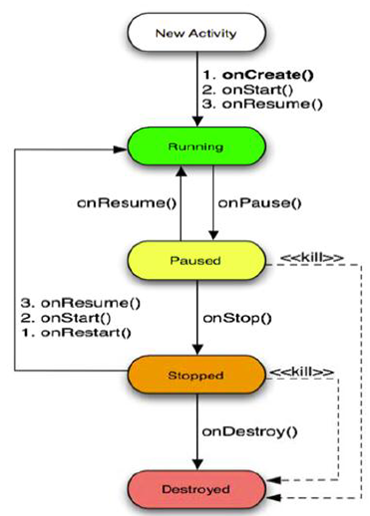
***5) Life Cycle States là gì?***

Với mỗi Activity thường vòng đời có 3 trạng thái sau:

1- **Running** (đang kích hoạt)

2- **Paused** (tạm dừng)

3- **Stopped** (dừng – không phải Destroyed)

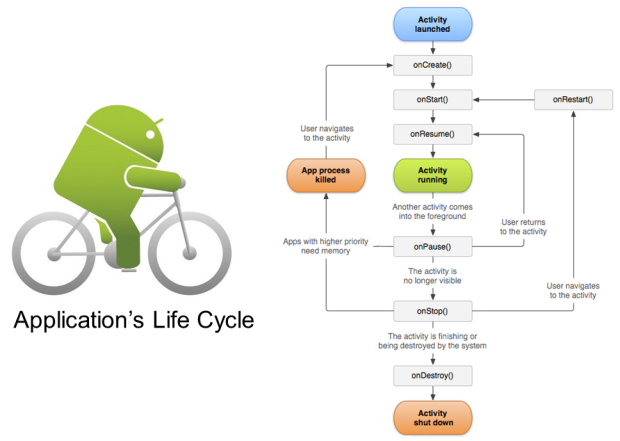
[](https://duythanhcse.files.wordpress.com/2013/03/5_life_cycle_3.png)

1- **Running** (đang kích hoạt): Khi màn hình là Foreground ( Activity nằm trên cùng ứng dụng và cho phép người sử dụng tương tác)

2- **Paused** (tạm dừng) : 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.

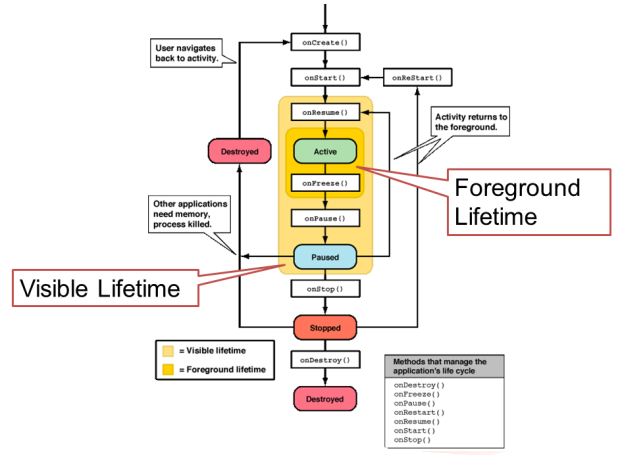
3- **Stopped** (dừng – không phải Destroyed): Activity mất focus và không nhìn thấy được (ví dụ bạn mở một Activity mới lên mà Full màn hình chẳng hạn). Trong trường hợp này nó có thể bị hệ thống “Xử” trong bất kỳ tình huống nào.

\*\*\* Như vậy cả **Paused** hay **Stopped** đều có khả năng bị **Destroyed** (hủy) khi bộ nhớ cần cho việc khác ưu tiên hơn.

[](https://duythanhcse.files.wordpress.com/2013/03/5_life_cycle_4.png)

– Trong vòng đời của ứng dụng Android bạn cần phần biệt 2 loại sau:

– **Visible Lifetime** và **Foreground Lifetime**

[](https://duythanhcse.files.wordpress.com/2013/03/5_life_cycle_5.png)

– **Visible Lifetime:**

+ sảy ra từ sau khi gọi onStart –> cho tới lúc gọi onStop : trong trường hợp này vẫn có thể thấy màn hình Activity (có thể tương tác khi nó là foreground, không tương tác được khi nó không phải foreground như đã giải thích ở trên)

**– Foreground Lifetime:**

+ Sảy ra từ khi gọi onResume –> cho tới lúc gọi onPause : trong suốt thời gian này Activity luôn nằm ở trên cùng và Ta có thể tương tác được với nó.

Trong phần câu 2 bài tập nhóm số 2, em sẽ nói rõ hơn về Visible Lifetime và Foreground Lifetime thông qua các demo của thầy.

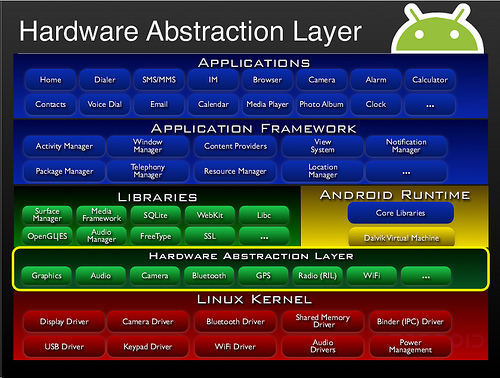
**– OnRestoreInstanceState:**

**Khi thực hiện xoay màn hình ứng dụng, ứng dụng sẽ gọi lại hàm OnRestoreInstanceSate và thực hiện Resume ứng dụng, vẽ lại ứng dụng mới tương thích với chiều màn hình.**

Nguồn tìm hiểu: https://duythanhcse.wordpress.com/2013/03/27/bai-tap-5-tim-hieu-vong-doi-cua-mot-ung-dung-android/

# 3. Tìm hiểu và giải thích Hardware Abstraction Layer (HAL) trong kiến trúc của Android?

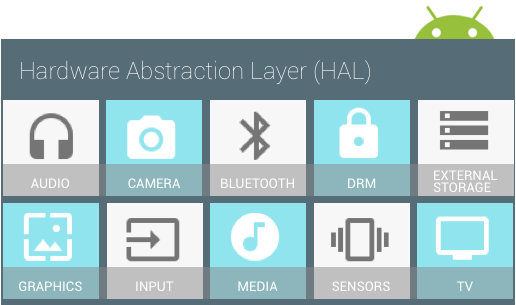
\*\*Tầng Hardware Abstraction Layer (HAL) trong kiến trúc Android:



Đây là ngăn có chức năng cung cấp các chuẩn giao tiếp phần mềm với phần cứng của thiết bị, giúp các ứng dụng trên Android có thể sử dụng được các phần cứng như camera, bluetooth… HAL chứa nhiều thư viện, nhiều module khác nhau, các thư viện này có chức năng điều khiển một phần cứng cụ thể.

Ví dụ: module camera sẽ chịu trách nhiệm điều khiển các tác vụ liên quan đến camera gắn trên điện thoại/ máy tính bảng, khi ứng dụng chúng ta viết cần camera thì module camera sẽ được gọi để giúp ứng dụng của chúng ta thực hiện chức năng đó.

Để các ứng dụng có thể truy cập đến phần cứng của thiết bị thì cần thông qua một lớp Hardware Abstraction Layer như sau:



* Lớp này cung cấp các giao diện cho phép các ứng dụng giao tiếp được với phần cứng, chúng có thể kết nối đến Kernel Driver thông qua việc gọi thư viện.
* Tầng này cũng cung cấp giao diện lập trình cho phép sử dụng nguồn tài nguyên Kernel.
* Sử dụng các phương thức open(), read(), write(), ioctl(), mmap(), close()...
* Android HAL cung cấp cách để load thư viện tự động như sau: id : Module ID yêu cầu load tự động, module - > return hw\_module\_t điểm của HMI (Hardware Module Information)