Phần I: Giới thiệu hệ điều hành Android

Hệ điều hành Android

Android là một hệ điều hành mở dành cho thiết bị di động dựa trên một phiên bản được chỉnh sửa của Linux. Hệ điều hành này ban đầu được phát triển bởi một công ty cùng tên là Android Inc. Vào năm 2005, với chiến lược thâm nhập vào thị trường di động đầy tiềm năng, Google đã mua lại Android và tham gia vào việc phát triển và hoàn thiện nó.

Google muốn Android trở thành một hệ điều hành mở và hoàn toàn miễn phí, tuy nhiên, hầu hết mã nguồn của Android đều được cho ra mắt dưới bản quyền Apache dành cho mã nguồn mở, vì vậy bất cứ ai muốn sử dụng Android phải tải về toàn bộ mã nguồn của Android. Hơn thế nữa, các nhà sản xuất (đặc biệt là nhà sản xuất phần cứng) có thể thêm vào phần mở rộng của họ vào Android và tùy biến nó để trở nên khác biệt so với sản phẩm của hãng khác. Kiểu mẫu phát triển như vậy đã khiến cho Android trở nên rất hấp dẫn và thu hút nhiều nhà sản xuất. Điều này đặc biệt có ý nghĩa đối với những công ty đang bị ảnh hưởng bởi hiện tượng iPhone của Apple, một sản phẩm rất thành công tạo nên một cuộc cách mạng trong ngành công nghiệp Smartphone bấy giờ. Trong số những công ty đó, có Motorola và Sony Ericsson, những công ty cũng nhiều năm phát triển hệ điều hành di động của riêng mình. Khi iPhone ra mắt, những công ty đó phải tìm nhiều cách để có thể làm hồi sinh những sản phẩm của mình. Và họ đã tìm ra câu trả lời, chính là Android. Họ quyết định tiếp tục phát triển phần cứng của riêng mình, và sử dụng Android như là hệ điều hành.

Một trong những thuận lợi khi lựa chọn Android, đó là nó thay đổi cách tiếp cận của sự phát triển ứng dụng. Các nhà phát triển (Developer) chỉ phải phát triển ứng dụng cho Android, và các ứng dụng của họ sẽ có thể chạy trên vô số các thiết bị, miễn là sử dụng hệ điều hành Android. Trong thế giới Smartphone, các ứng dụng là phần quan trọng nhất trong dây chuyền thành công của sản phẩm. Các nhà sản xuất thiết bị coi Android là hi vọng lớn nhất của họ để có thể cạnh tranh với iPhone của Apple, một thiết bị đã có quá nhiều những ứng dụng lớn.

1. Các phiên bản Android

Kể từ khi phát hành chính thức, Google đã cho ra mắt rất nhiều phiên bản khác nhau của Android, có thể kể đến thời điểm hiện tại những phiên bản đã được ra mắt như dưới đây:

Phiên bản	Tên mã
1.1	
1.5	Cupcake
1.6	Donut
2.0/ 2.1	Éclair

2.2	Froyo
2.3	Gingerbread
3.0/ 3.1/ 3.2	Honeycomb
4.0	Ice Cream Sandwich
4.1	Jelly Bean
4.2	Jelly Bean
4.3	Jelly Bean
4.4	KitKat

Kể từ phiên bản 1.5, các phiên bản Android lần lượt được đặt tên theo các loại bánh, theo thứ tự của bảng chữ cái. Phiên bản mới nhất vừa được ra mắt là phiên bản 4.4 với tên mã là KitKat.

Phiên bản KitKat ra mắt vào tháng 11/2013, từng bước hiện thực ước muốn của Google, đó là phổ biến hệ điều hành phiên bản mới nhất cho các dòng điện thoại có cấu hình ở mức trung bình. Google cho biết phiên bản KitKat sẽ có thể điều tiết lượng RAM tùy thuộc vào phần cứng của thiết bị, và KitKat có thể chạy trên những thiết bị có RAM là 512MB.

2. Các tính năng của Android

Vì Android là mã nguồn mở và hoàn toàn miễn phí đối với các nhà sản xuất để có thể tùy biến, nên không có một tùy chỉnh cấu hình phần cứng và phần mềm cố định nào. Tuy nhiên, bản thân Android hỗ trợ các tính năng sau:

- Lưu trữ: Sử dụng SQLite để lưu trữ dữ liệu.
- Kết nối: Hỗ trợ GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (bao gồm A2DP và AVRCP), Wi-Fi, LTE và WiMAX.
- Tin nhắn: Hỗ trợ MMS và SMS.
- Trình duyệt: Dựa trên mã nguồn mở WebKit, cùng với Engine V8 JavaScript của Chrome.
- Hỗ trợ đa phương tiện: Hỗ trợ các chuẩn media: H.263, H.264 (ở định dạng 3GP hoặc MP4), MPEG-4 SP, AMR, AMR-WB (ở định dạng 3GP), AAC, HE-AAC (ở định dạng MP4 hoặc 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF và BMP.

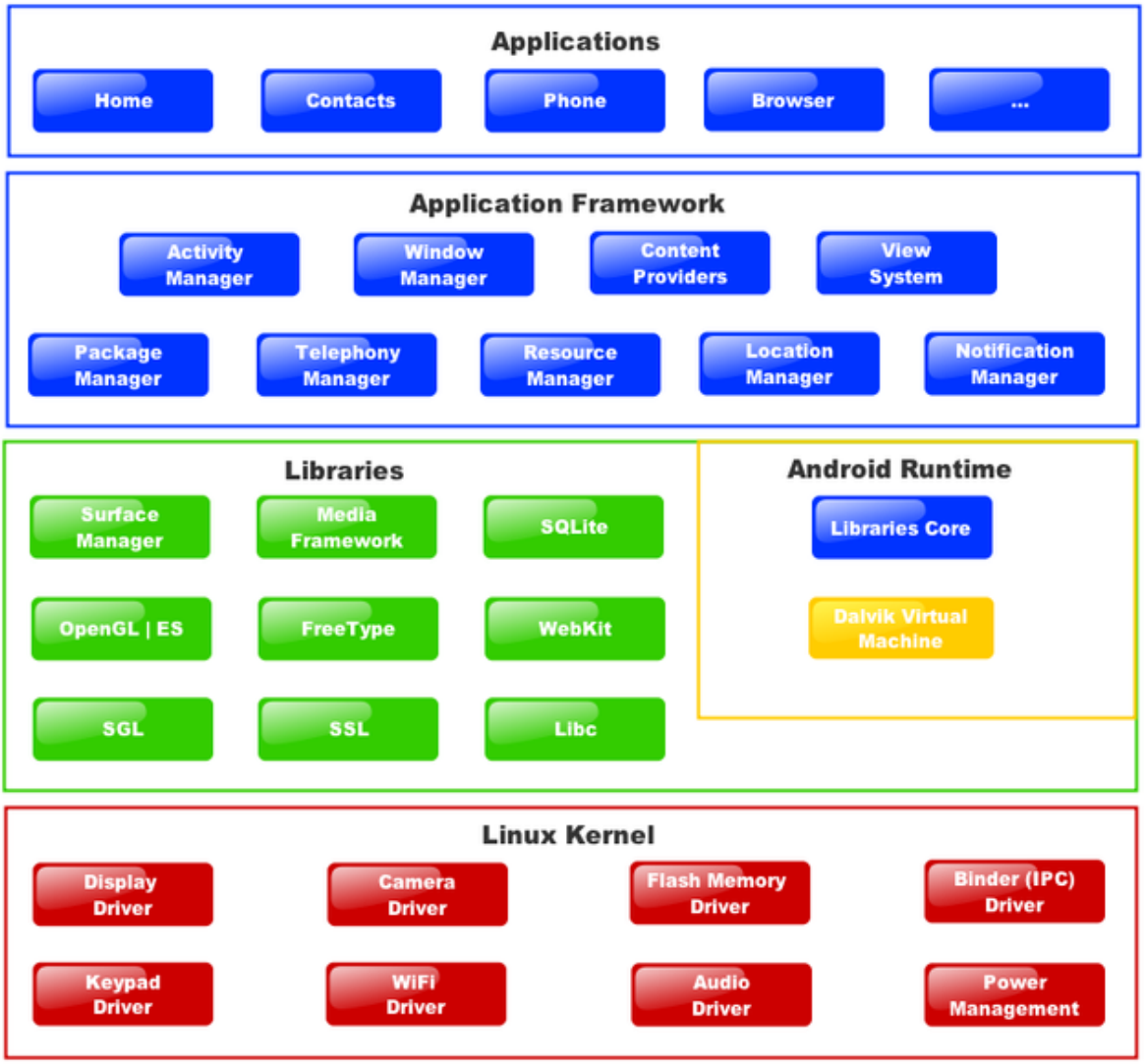
- Hỗ trợ phần cứng: Cảm biến gia tốc (Accelerometer Sensor), Camera, La bàn số (Digital Compass), Cảm biến tiệm cận (Proximity Sensor) và GPS.
- Đa chạm: Hỗ trợ màn hình đa chạm.
- Đa nhiệm: Hỗ trợ những ứng dụng đa nhiệm.
- Hỗ trợ Flash: Android 2.3 hỗ trợ Flash 10.1.
- Phát sóng (Tethering): Hỗ trợ chia sẻ kết nối Internet như là một trạm phát không dây (Wireless hotspot).

3. Cấu trúc của Android

Để có thể hiểu cách thức vận hành của Android, ta sẽ cùng tìm hiểu các lớp (layers) mà tạo nên hệ điều hành Android.

Hệ điều hành Android được chia thành năm phần với bốn lớp chính:

- **Linux Kernel**: đây là kernel (nhân) mà Android dựa trên. Lớp này bao gồm toàn bộ những drivers thiết bị cấp thấp cho các thành phần phần cứng đa dạng của thiết bị Android.
- **Libraries**: Lớp này bao gồm toàn bộ code mà cung cấp những tính năng chính của Android. Ví dụ thư viện SQLite cung cấp hỗ trợ cơ sở dữ liệu mà nhờ đó ứng dụng có thể sử dụng để lưu trữ dữ liệu, hoặc thư viện WebKit cung cấp những tính năng để duyệt web.
- **Android runtime**: Ở cùng lớp với Libraries, Android runtime cung cấp một bộ thư viện lõi mà giúp các nhà phát triển có thể viết ứng dụng Android bằng ngôn ngữ lập trình Java. Android runtime còn bao gồm máy ảo Dalvik, giúp cho mỗi ứng dụng Android có thể chạy một process riêng bằng một Instance của máy ảo Dalvik (các ứng dụng Android được biên dịch thành các file thực thi Dalvik). Dalvik là một máy ảo được thiết kế đặc biệt cho Android và được tối ưu cho các thiết bị dùng pin với bộ nhớ và CPU hạn chế.
- **Android Framework**: Chỉ ra những khả năng của hệ điều hành Android, từ đó các nhà phát triển Android có thể sử dụng chúng trong những ứng dụng của mình.
- **Applications**: Ở lớp trên cùng này, ta có thể thấy những ứng dụng đi kèm với thiết bị Android (Danh bạ, Trình duyệt, vv) cũng như những ứng dụng ta download và cài đặt từ Market. Tất cả những ứng dụng mà ta viết đều nằm ở lớp này.


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
Libc
Android Runtime
Libraries Core
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Driver
Power Management

4. The Android Market

Giống như đã nói ở trên, yếu tố chính quyết định đến sự thành công của một nền tảng di động chính là những ứng dụng hỗ trợ nó. Từ sự thành công của iPhone có thể thấy rất rõ ràng rằng những ứng dụng có vai trò vô cùng quan trọng trong việc quyết định một nền tảng mới sẽ tồn tại hay bị diệt vong. Thêm vào đó, việc làm cho những ứng dụng này đến với những người dùng phổ thông là hết sức quan trọng.

Vì vậy, vào tháng 8/2008, Google thông báo sẽ mở ra Android Market, một kho ứng dụng online cho những thiết bị Android, và mở cửa vào tháng 10/2008. Sử dụng ứng dụng Market được cài đặt sẵn, người dùng có thể download những ứng dụng của bên thứ ba một cách đơn giản. Market hỗ trợ cả ứng dụng miễn phí và trả phí, mặc dù một vài ứng dụng trả phí chỉ dành cho những người dùng ở những quốc gia nhất định vì lí do pháp lí.

5. Những công cụ cần thiết

Để có thể phát triển ứng dụng Android, ta cần download những công cụ cần thiết và các bộ SDK.

Để lập trình Android, ta có thể sử dụng một máy tính chạy hệ điều hành Windows, MAC hoặc Linux. Tất cả những công cụ đều miễn phí và có thể download trực tiếp từ Web.

Trước khi download Android SDK, ta cần cài đặt Java JDK, vì Android SDK sử dụng Java SE Development Kit (JDK). Ta có thể download dễ dàng trên trang chủ của Oracle: www.oracle.com/technetwork/java/javase/downloads/index.html

Tiếp theo sau khi đã download và cài đặt JDK, ta sẽ download và cài đặt phần mềm quan trọng nhất, chính là Android SDK. Android SDK bao gồm debugger, các thư viện, bộ mô phỏng emulator, các tài liệu, code mẫu và các bài hướng dẫn.

Ta có thể download Android SDK trên website dành cho nhà phát triển của Android: http://developer.android.com/sdk/index.html

Sau khi download xong Android SDK, ta tiến hành cài đặt, lưu ý khi chọn địa chỉ lưu bộ SDK.

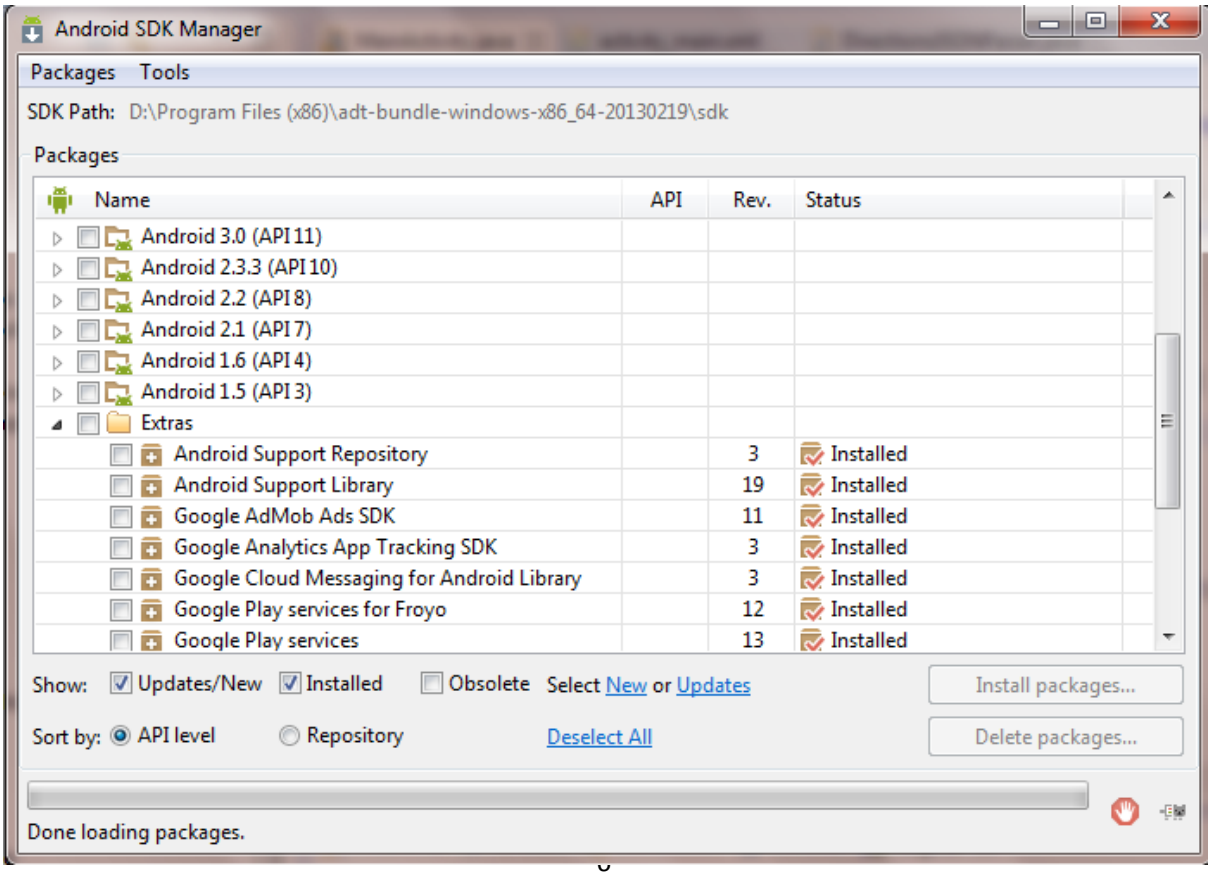
6. Tùy chỉnh Android SDK Manager

Android SDK Manager là một phần mềm quản lý các phiên bản Android SDK hiện tại được cài đặt trong máy. Khi khởi động nó sẽ cho thấy một danh sách các mục và các mục đó đã được cài đặt hay chưa.

Đánh dấu vào các mục ta cần download, thường thì bộ SDK download từ ở trên sẽ chỉ gồm Platform phiên bản Android mới nhất, nếu muốn lập trình trên những phiên bản Android thấp hơn, ta phải chọn download các Platform còn thiếu từ SDK Manager.

Đề tài luận văn là "Lập trình xe tự hành dựa trên Google Maps API", nên phần quan trọng không thể thiếu là bộ SDK phải hỗ trợ Google Maps API. API này nằm trong Google Play Service, nên ta phải download mục này thì mới có thể lập trình được.

Hình ảnh của Android SDK Manager:



7. Eclipse

Bước tiếp theo là cài đặt môi trường phát triển tích hợp (IDE) để phát triển ứng dụng Android. Và Google khuyến khích sử dụng Eclipse, một IDE hỗ trợ lập trình nhiều ngôn ngữ phổ biến như Java, Ada, C, C++, COBOL, Python, …

Để lập trình Android, ta nên download Eclipse IDE for Java EE Developers. Ta phải chọn hệ điều hành phù hợp để download phiên bản Eclipse phù hợp cho mình. Trong trường hợp này là Windows 64 bit.

Các bước download và cài đặt khá đơn giản. Chỉ cần giải nén file download được là có thể sử dụng được Eclipse bằng cách mở file thực thi eclipse.exe.

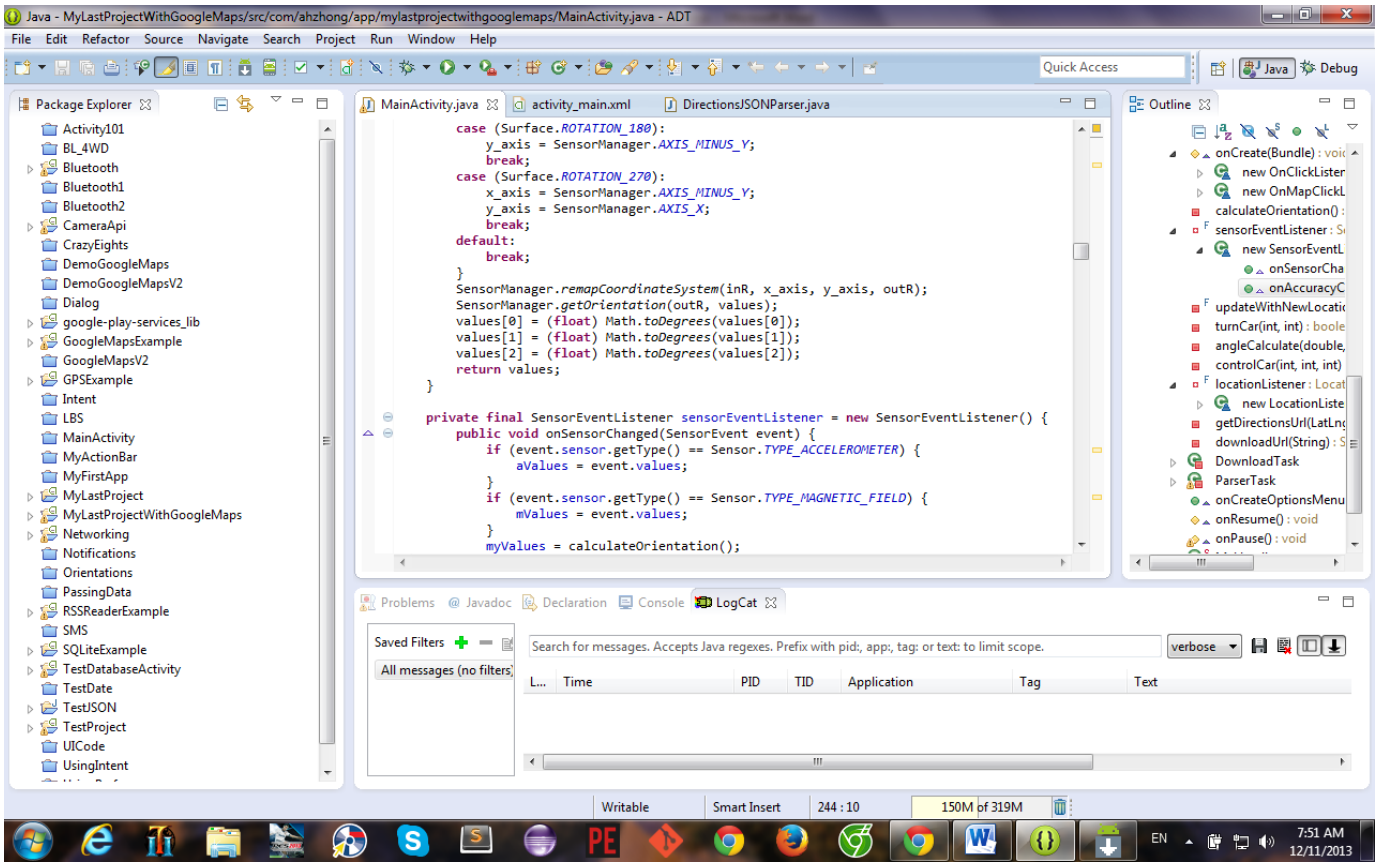
Đối với những bản Android SDK sau này (từ 4.2 trở đi), thì Google đã giúp tích hợp Eclipse vào bộ Android SDK, khi chúng ta giải nén Android SDK thì cũng có sẵn một thư mục Eclipse, ta không cần phải download thêm ở ngoài nữa.

Thêm vào đó, việc tích hợp sẵn Eclipse vào bộ cài đặt Android SDK cũng giảm nhiều công sức cho người dùng, vì phiên bản Eclipse này đã được cài đặt sẵn phần mở rộng ADT, mà ngày xưa người dùng phải tự cài đặt thêm khi cài đặt Eclipse.

Tiện ích mở rộng ADT sẽ giúp chúng ta:

- Tạo project Android mới.
- Tiếp cận với các công cụ để tiếp cận bộ Emulator hoặc thiết bị Android của chúng ta.
- Biên dịch và debug chương trình Android.
- Export ứng dụng Android (file APK).
- Tạo chứng chỉ số cho chữ kí mã hóa cho file APK.

Hình ảnh của Eclipse:

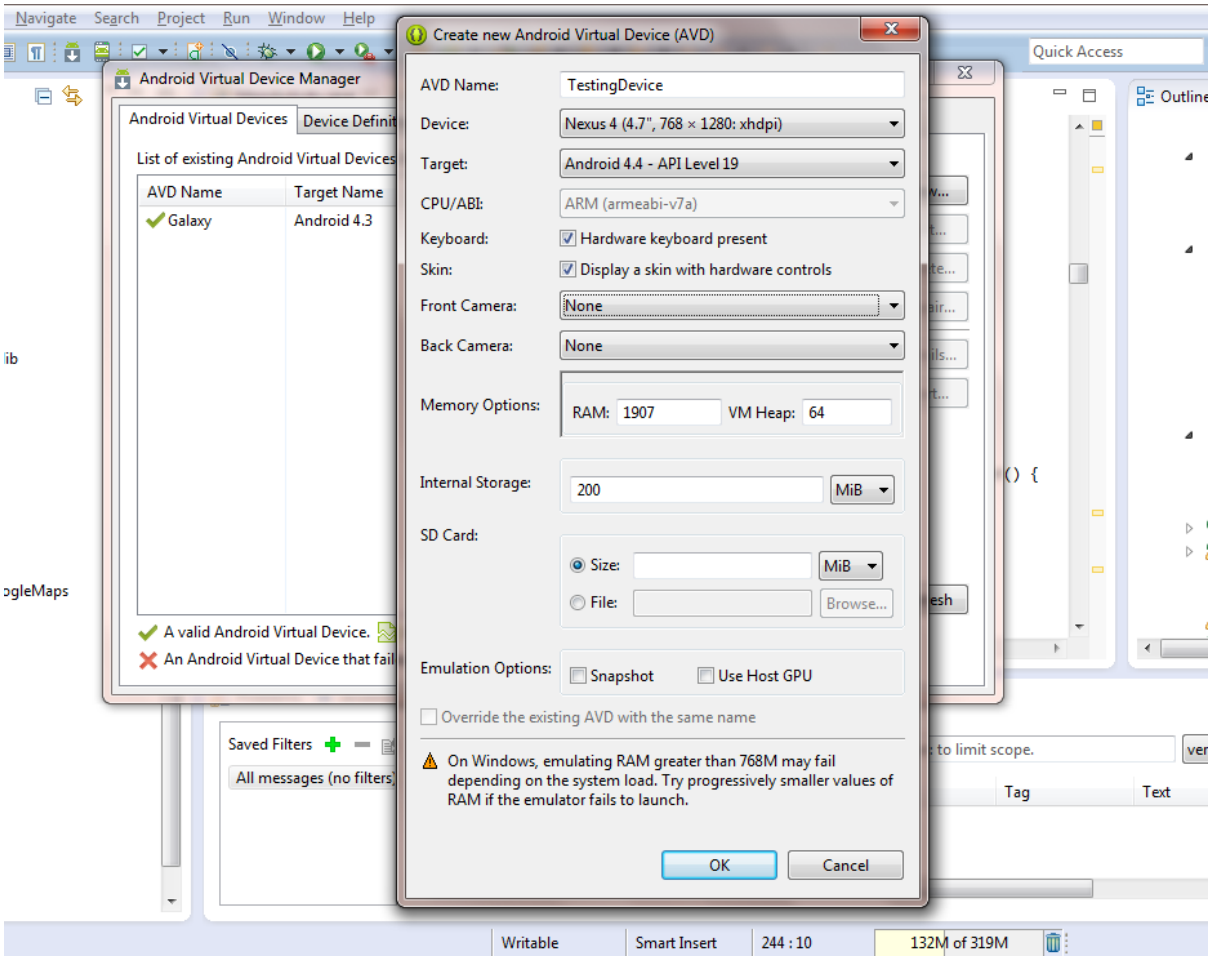


8. Tạo thiết bị ảo Android (Android Virtual Devices – AVD)

Một phần không thể thiếu của bộ công cụ lập trình ứng dụng Android, chính là thiết bị ảo Android – AVD. AVD được Google tích hợp vào bộ Android SDK nhằm giúp người dùng có thể chạy thử các ứng dụng của họ. AVD có thể mô phỏng hoạt động của một thiết bị thực. Mỗi AVD có một cấu hình phần cứng, file hệ thống cũng như bộ nhớ (vd như SD card).

Ta có thể tạo nhiều AVD để có thể chạy thử chương trình trên nhiều cấu hình phần cứng và phần mềm khác nhau. Việc chạy thử này rất quan trọng để kiểm tra ứng dụng có hoạt động đúng trên nhiều thiết bị có cấu hình khác nhau hay không.

Trong Eclipse, ta tạo AVD bằng cách vào Window -> AVD Manager, chọn New ở cửa sổ Android Virtual Device Manager, cửa sổ mới hiện lên sẽ cho ta tùy chỉnh AVD.



Như ở hình ví dụ ở trên, ta đang tạo một AVD với tên là TestingDevice, chọn mô phỏng điện thoại Nexus 4, hệ điều hành là Android 4.4, các thông số RAM và bộ nhớ trong được set như trên hình.

Khi lập trình ứng dụng thì ta nên tạo nhiều Android với nhiều cấu hình phần cứng và phần mềm khác nhau để kiểm tra sự tương thích và hoạt động của ứng dụng.

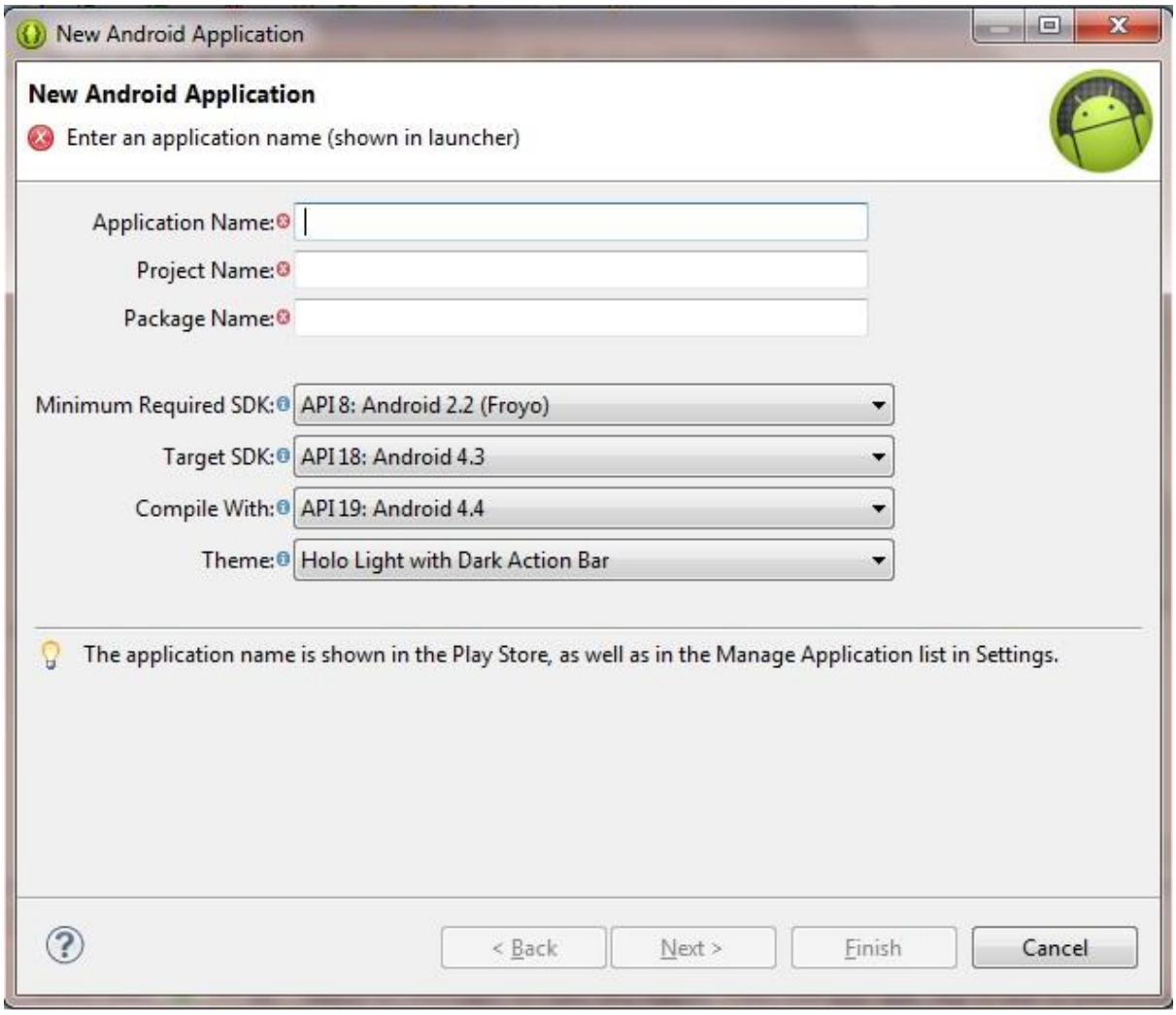
Tuy nhiên, AVD cũng có một vài nhược điểm:

- Vì là một thiết bị ảo Android, nên dẫn đến AVD chạy không được mượt như thiết bị thực, thậm chí đáp ứng rất chậm, gây khó khăn trong việc chạy thử ứng dụng.
- AVD không thể mô phỏng những ứng dụng đòi hỏi sử dụng phần cứng của điện thoại, ví dụ như Bluetooth, Sensor, …

Trong phạm vi của luận văn, rõ ràng không thể sử dụng AVD để mô phỏng, do luận văn sử dụng nhiều tính năng phần cứng của một thiết bị thực như Bluetooth và các Sensor, vì vậy chỉ có thể chạy trên một thiết bị thực mới có thể kiểm tra được ứng dụng.

9. Tạo một ứng dụng Android trong Eclipse

Trong Eclipse, mở File -> New -> Android Application Project.



Sau đó nhập các field yêu cầu:

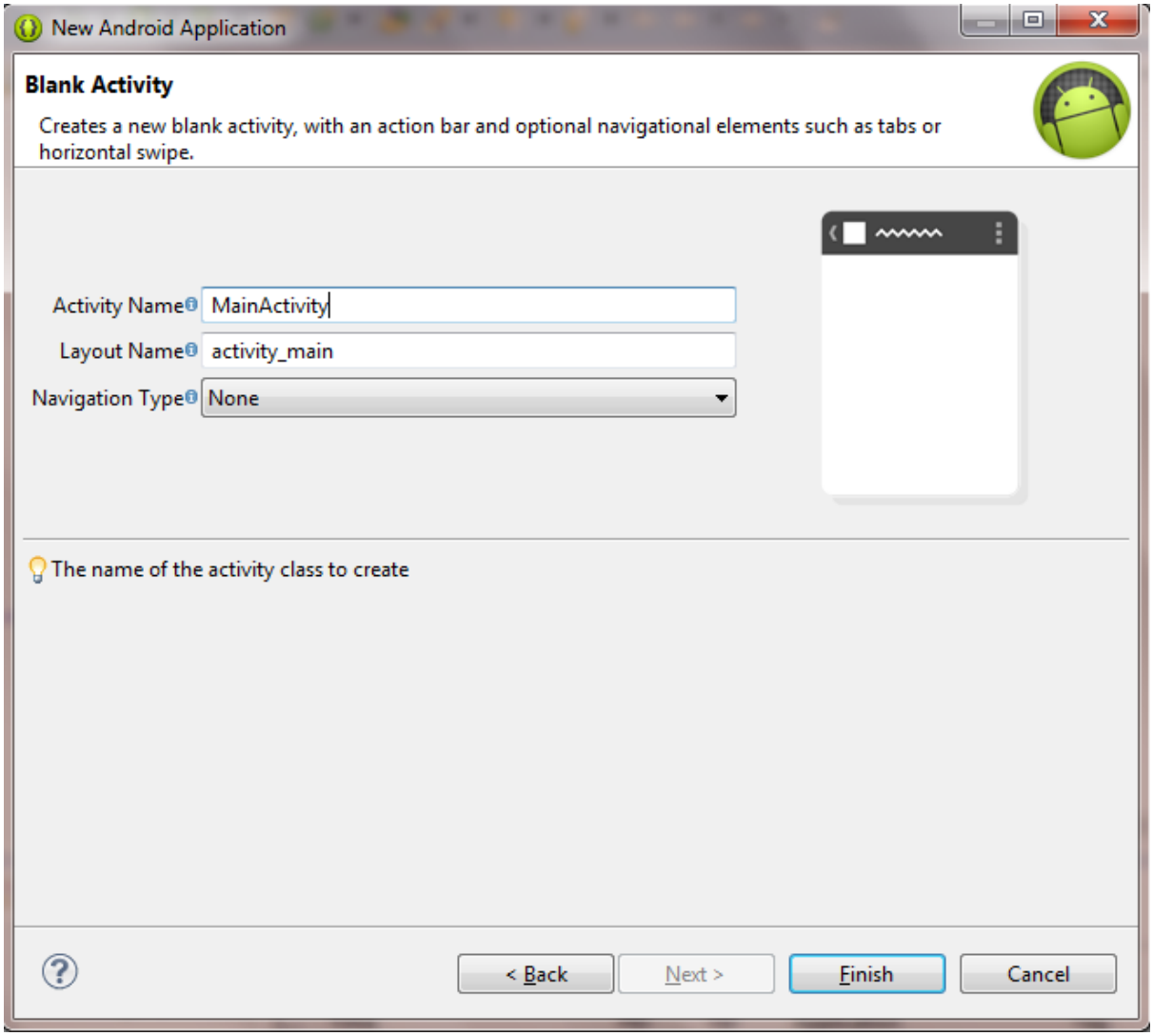
- Application Name: tên của ứng dụng, là tên mà người dùng nhìn thấy, nên có thể có khoảng trắng và mô tả ứng dụng.
- Project Name: tên của project trong Eclipse, và bắt buộc đặt tên theo quy định của một project trong Eclipse (không được có khoảng trắng và các kí tự đặc biệt).

- Package Name: tên của package chứa các Class của project. Trong phạm vi luận văn này, package name còn dùng để lấy API key mà Google cung cấp cho các ứng dụng sử dụng Google Maps API.
- Minimum Required SDK: nền tảng Android thấp nhất mà ứng dụng hỗ trợ.
- Target SDK: là nền tảng Android mà ứng dụng hướng đến, có thể coi là nền tảng Android cao nhất mà ứng dụng hỗ trợ.
- Compile with: là nền tảng Android dùng để biên dịch ứng dụng, thường chọn bằng với Target SDK.

Sau khi thiết lập xong ta chọn Next, ở hai bước tiếp theo là các công đoạn lựa chọn các tùy chọn (như workspace, icon), ta có thể để mặc định hai bước này và bấm Next để đến bước tiếp theo.

Bước tiếp theo là Create Activity, Activity có thể hiểu là một Class để định nghĩa hành vi của ứng dụng khi tương tác với người dùng, có thể coi đơn giản Activity là một cửa sổ để người dùng có thể tương tác thông qua các đối tượng. Ở mục tiếp theo bên dưới sẽ nói thêm về Activity trong một ứng dụng Android.

Cửa sổ này ta có thể để mặc định dấu check ở Blank Activity, nghĩa là Eclipse sẽ tạo cho ta một Activity trống. Ta bấm Next qua bước tiếp theo.

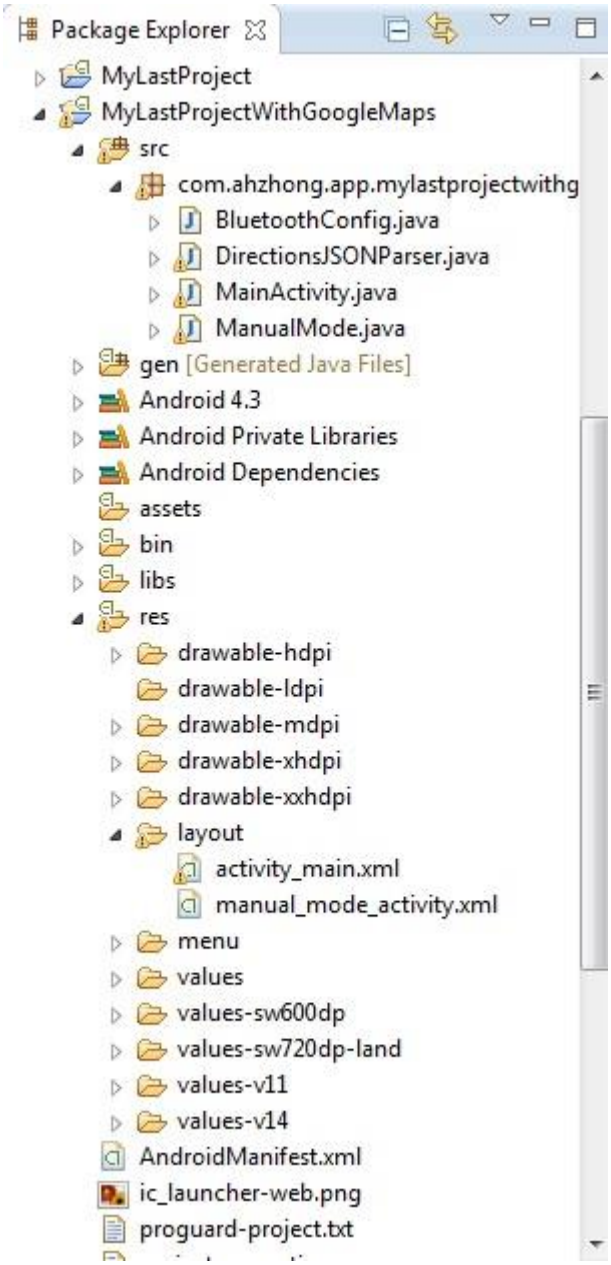


Ở cửa sổ tiếp theo, ta sẽ điền vào hai field:

- Activity Name: tên của Activity, thông thường trong một chương trình Android sẽ có rất nhiều Activity khác nhau, tương ứng với những cửa sổ mà người dùng tương tác, Activity Name nên đặt theo behavior của nó (ví dụ như Activity của mode manual nên đặt tên là ManualModeActivity).
- Layout Name: vì Activity định nghĩa các Class dùng để tương tác với người dùng, nên ứng với mỗi Activity sẽ có một Layout dùng để mô tả giao diện cho nó, ta cũng nên đặt tên Layout có liên quan đến Activity mà nó mô tả.

10. Cấu trúc của một ứng dụng Android

Chúng ta sẽ cùng xem xét những file và folder được tạo ra trong quá trình phát triển ứng dụng Android:



- Src: là thư mục chứa các file mã nguồn .java cho project, ta sẽ thấy các file này nằm trong package mà ta đặt tên lúc đầu, ngoài các Activity ra ta có thể tạo thêm nhiều Class thông thường khác để định nghĩa các Config cần thiết cho project.
- Gen: Chứa file R.java, một file tạo bởi bộ biên dịch để tham chiếu đến tất cả tài nguyên trong project. Ta không nên chỉnh sửa file này, vì tất cả các tài nguyên sẽ được tự động biên dịch vào file này và ta có thể tham chiếu đến nó.
- Android 4.3: chứa file android.jar, chứa toàn bộ các thư viện cần thiết cho ứng dụng Android.
- Assets: chứa những asset dùng bởi ứng dụng, như HTML, database hoặc file text, …
- Bin: folder này chứa các file được build bởi ADT (Android Development Tool), và đây cũng là nơi tạo ra file .apk (Android Package). File .apk là file thực thi của Android, nó chứa mọi thứ cần để chạy được ứng dụng Android. Có thể coi file .apk tương đương với file .exe của hệ điều hành Windows.
- Res: folder này chứa tất cả tài nguyên (resources) của project. Folder này có một vài sub-folder như drawable - <resolution> dùng để chứa các file ảnh, layout dùng để chứa các file layout, values dùng để chứa các giá trị (như string, array, …)
- AndroidManifest.xml: đây là file khai báo của project. Ta sẽ phải khai báo toàn bộ các permission (ví dụ như Internet permission, Bluetooth permission, …) và khai báo toàn bộ các Activity dùng trong chương trình.

Hình ảnh ví dụ của AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.HelloWorld"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".HelloWorldActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Trong một file AndroidManifest.xml, trước hết vì đây là định dạng file XML, nên nó sẽ có cấu trúc bao gồm các thẻ <element></element>, và bên trong có những element khác lồng vào để định nghĩa các element lớn hơn.

- AndroidManifest.xml trước hết khai báo Package Name của project đang thực hiện trong thuộc tính package của thẻ <manifest>.
- Tiếp theo là khai báo mã phiên bản (version code), khai báo này nhằm mục đích để xác định xem một ứng dụng đã được cài sẵn có cần nâng cấp hay không.
- Tiếp đến là tên phiên bản (version name), đây là tên được hiển thị tới người dùng, thường đặt theo cú pháp là <major>.<minor>.<point>.
- Tiếp đến là thuộc tính minSdkVersion của thẻ <uses-sdk>, dùng để khai báo phiên bản Android thấp nhất mà ứng dụng hỗ trợ.
- Ứng dụng sẽ lấy ảnh ic_launcher.png nằm ở thư mục drawables để làm icon.
- Trong ví dụ này, chương trình chỉ có một Activity nên ta chỉ khai báo một Activity này trong file AndroidManifest.xml

Khi ta khai báo trong file Manifest và tạo các folder, các file, thì lập tức Eclipse sẽ tự động build file R.java. Ta cùng xem ví dụ một file R.java như bên dưới:

```
public final class R {
    public static final class attr {
    }
    public static final class dimen {
        /**  Default screen margins, per the Android Design guidelines.

         Customize dimensions originally defined in res/values/dimens.xml (such as
         screen margins) for sw720dp devices (e.g. 10" tablets) in landscape here.

         */
        public static final int activity_horizontal_margin=0x7f040000;
        public static final int activity_vertical_margin=0x7f040001;
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f080000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class menu {
        public static final int my_last_project=0x7f070000;
    }
    public static final class string {
        public static final int action_settings=0x7f050001;
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050002;
    }
```

Như file ví dụ ở trên, khi ta thêm các file hoặc folder vào project, Eclipse sẽ tự động chỉnh sửa file R.java, và chúng ta không nên chỉnh sửa file này.

Giới thiệu Activity

Như đã giới thiệu ở trên, một Activity là một cửa sổ chứa các giao diện người dùng (User Interface) của ứng dụng Android. Một ứng dụng có thể không có hoặc có nhiều Activity khác nhau. Thông thường, các ứng dụng có một hoặc nhiều Activity, và mục đích chính của Activity là để tương tác với người dùng.

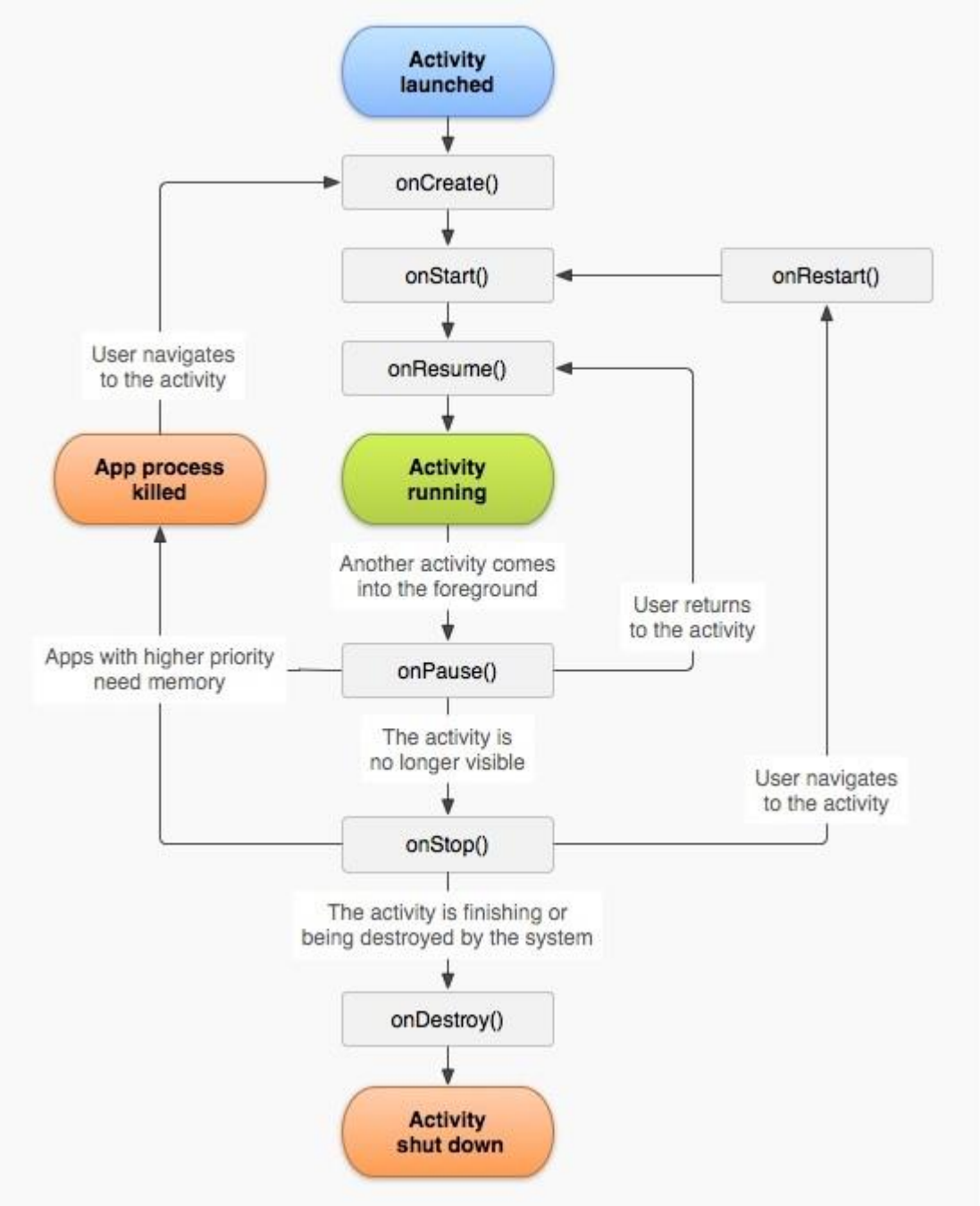
1. Life cycle của một Activity

Từ lúc Activity xuất hiện trên màn hình cho đến khi bị che khuất đi, nó trải qua nhiều giai đoạn, thường được gọi là Life Cycle của Activity. Hiểu rõ Life Cycle của Activity là một điều quan trọng để đảm bảo rằng ứng dụng của chúng ta hoạt động đúng. Các Activity trong hệ thống được quản lý theo các "Activity Stack". Khi một Activity mới được bắt đầu, nó sẽ được đặt trên cùng của Stack, và trở thành Activity được thực thi, khi đó Activity trước đó sẽ nằm bên dưới trong Stack, và chỉ xuất hiện trở lại màn hình chính khi Activity mới thoát ra.

Một Activity có bốn trạng thái chính:

- Nếu một Activity ở trên màn hình chính (ở trên cùng của Stack), ta nói Activity đó đang "Active" hoặc "Running".
- Nếu một Activity không còn ở màn hình chính nữa nhưng vẫn nhìn thấy được (một Activity không full-screen mới đã xuất hiện ở trên cùng của Stack), ta nói nó đã bị "Paused". Một Activity bị Paused thực chất vẫn đang hiện hữu (nó giữ nguyên các trạng thái, thông tin của các thành viên và vẫn gắn với Window Manager), nhưng khi trạng thái thiếu bộ nhớ thì nó sẽ bị triệt tiêu.
- Khi một Activity bị che lấp hoàn toàn bởi một Activity mới, ta nói nó bị "Stopped". Nó vẫn giữ nguyên các trạng thái và thông tin của thành viên nhưng cửa sổ của nó đã bị che lấp hoàn toàn và nó dễ bị triệt tiêu bởi hệ thống khi hệ thống cần bộ nhớ.
- Nếu một Activity bị "Paused" hoặc "Stopped", nó có thể bị hệ thống triệt tiêu để giải phóng bộ nhớ (bằng cách bắt buộc Activity hoàn thành, hoặc triệt tiêu các process của nó). Khi Activity này được hiển thị trở lại, nó phải được khởi động lại hoàn toàn và khôi phục lại các trạng thái trước đây.

Biểu đồ sau diễn tả các trạng thái quan trọng của Activity. Các hình chữ nhật biểu thị các hàm (Callback Method) mà ta có thể lập trình khi Activity chuyển trạng thái. Những hình oval có màu là các trạng thái chính của các Activity.



Toàn bộ Life Cycle của một Activity được định nghĩa bởi các hàm bên dưới. Ta hoàn toàn có thể Override lại các hàm này để ứng dụng có thể làm việc như mong muốn trong quá trình chuyển trạng thái. Tất cả Activity sẽ phải chạy hàm onCreate(Bundle) để setup ban đầu, và thông thường cả hàm onPause() để xác thực các thay đổi đến dữ liệu hoặc chuẩn bị dừng tương tác với người dùng.

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Và thông thường sự chuyển dịch giữa các Activity có thể được mô tả như bên dưới:

Method	Mô tả	Có thể triêt tiêu?	Tiếp theo
`onCreate()`	Được gọi khi Activity được khởi tạo lần đầu tiên. Đây là nơi ta thực hiện những setup đầu tiên: tạo các giao diện, … Hàm này cũng cung cấp một Bundle lưu giữ trạng thái lúc trước (nếu có). Luôn theo sau bởi `onStart()`.	Không	`onStart()`
`onRestart()`	Được gọi sau khi Activity bị Stopped, trước khi nó được bắt đầu lần tiếp theo. Luôn theo sau bởi `onStart()`.	Không	`onStart()`
`onStart()`	Được gọi khi Activity được hiển thị trước người dùng. Theo sau bởi `onResume()` nếu Activity hiển thị ở màn hình chính, hoặc `onStop()` nếu nó bị che đi.	Không	`onResume()` or `onStop()`
`onResume()`	Được gọi khi Activity bắt đầu tương tác với người dùng. Khi này Activity ở trên cùng của Stack và sẽ nhận input từ phía người dùng. Luôn theo sau bởi `onPause()`.	No	`onPause()`
`onPause()`	Được gọi khi hệ thống chuẩn bị bắt đầu khôi phục một Activity trước đó. Đây là lúc ta lưu các dữ liệu, ngừng các luồng có thể tiêu tốn tài nguyên của thiết bị. Theo sau bởi `onResume()` nếu Activity trở lại, hoặc `onStop()` nếu Activity bị che khuất.	**Pre-**`HONEYCOMB`	`onResume()` or `onStop()`

`onStop()`	Được gọi khi Activity không còn được người dùng nhìn thấy, bởi vì một Activity khác đã được khôi phục và che khuất. Điều này có thể do một Activity mới xuất hiện, hoặc một Activity cũ khôi phục lại. Được theo sau bởi `onRestart()` nếu Activity trở lại tương tác với người dùng hoặc `onDestroy()` nếu Activity này không quay lại.	**Có**	`onRestart()` or `onDestroy()`
`onDestroy()`	Hàm cuối cùng mà ta gọi trước khi Activity bị Destroy. Điều này xảy ra khi Activity đã hoàn thành (khi ai đó gọi hàm finish() hoặc hệ thống tạm thời triệt tiêu để tiết kiệm bộ nhớ).	**Có**	*nothing*

2. Intent và bắt đầu một Activity mới

Khi muốn bắt đầu một Activity mới, ta sẽ gọi hàm `startActivity(Intent)`. Activity được gọi sẽ nằm ở trên cùng của Stack. Hàm này cần một tham số, đó là một đối tượng "Intent", dùng để mô tả Activity mới.

Một Intent thực chất có thể coi như là một thứ kết dính các Activity lại với nhau, dù có thể các Activity này không cùng nằm trong một ứng dụng. Thông thường ta dùng để gọi đến các Activity trong cùng một ứng dụng, hoặc các Activity mặc định của hệ thống (ví dụ như Trình duyệt, Bản đồ, Bluetooth, …).

Đôi khi ta muốn nhận một kết quả trả về từ Activity mà ta gọi (khi nó kết thúc và trả Activity cũ về vị trí trên cùng của Stack). Ví dụ như ta gọi một Activity của hệ thống để xin phép bật Bluetooth, và sau đó ta muốn biết là người dùng có cho phép hay không, bằng cách đọc kết quả mà Activity đó trả về. Để làm được điều đó, thay vì gọi hàm `startActivity(Intent)` thì ta sẽ gọi hàm `startActivityForResult(Intent, int)`, với tham số thứ hai để nhận diện sự gọi đến Activity đó. Kết quả trả về sẽ gọi hàm `onActivityResult(int, int, Intent)`.

Ta xét một ví dụ như dưới đây:

```
public class MyActivity extends Activity {
    ...

    static final int PICK_CONTACT_REQUEST = 0;

    protected boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
            // When the user center presses, let them pick a contact.
            startActivityForResult(
                new Intent(Intent.ACTION_PICK,
                new Uri("content://contacts")),
                PICK_CONTACT_REQUEST);
           return true;
        }
        return false;
    }

    protected void onActivityResult(int requestCode, int resultCode,
            Intent data) {
        if (requestCode == PICK_CONTACT_REQUEST) {
            if (resultCode == RESULT_OK) {
                // A contact was picked.  Here we will just display it
                // to the user.
                startActivity(new Intent(Intent.ACTION_VIEW, data));
            }
        }
    }
}
```

Ở ví dụ ở trên, ta thấy hàm `startActivityForResult(Intent, int)` gửi một Intent mặc định là ACTION_PICK, với tham số thứ hai là PICK_CONTACT_REQUEST, đây là một số Integer mà người dùng tự định nghĩa, để khi nhận kết quả trả về trong hàm `onActivityResult(int, int, Intent)`, ta có thể so sánh xem kết quả đó được trả về từ Activity nào (trong trường hợp ta gọi nhiều Activity), như trong ví dụ, ta có một câu lệnh dùng để kiểm tra xem có phải kết quả được trả về từ Activity ta mong muốn hay không, sau đó ta sẽ kiểm trả resultCode, có hai kết quả thông thường là RESULT_OK và RESULT_CANCELED cho biết kết quả thực thi của Activity.

3. Giao diện người dùng

Trong các chương trình Android, một Activity thường có mục đích chính là để tương tác với người dùng. Vì vậy Android cung cấp cho chúng ta một Class Activity để ta có thể thừa kế và tùy biến cho các Activity của riêng mình.

Một trong những điều Class Activity mang lại cho chúng ta, là nó sẽ cung cấp hàm để tạo nên một cửa sổ cho người dùng, từ cửa sổ đó chúng ta có thể tùy biến Giao Diện Người Dùng (User Interface – UI). Hàm mà Class Activity cung cấp cho chúng ta là hàm "setContentView(View)". Trong đó View là một Class được định nghĩa sẵn của hệ điều hành Android, mô tả thuộc tính cho các đối tượng View mà người dùng sẽ tương tác (có thể kể ra như Button, TextView, EditText, …).

Các Activity thường được hiển thị lên màn hình theo kích thước full-screen để có thể dễ dàng tương tác với người dùng, ta hoàn toàn có thể thay đổi các thể hiện mặc định này.

Và để định nghĩa giao diện người dùng (UI), ta sẽ sử dụng file XML (Layout Name) nằm trong thư mục res/layout.

Ví dụ của một file XML định nghĩa layout của một Activity:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</LinearLayout>
```

Ở file này ta định nghĩa layout của Activity là kiểu LinearLayout, với các thuộc tính độ dài và độ rộng là "fill_parent", nghĩa là layout sẽ tràn ra toàn bộ phần nhìn thấy của Activity. Layout có một đối tượng TextView, với độ rộng là "fill_parent", độ cao là "wrap_content", nghĩa là sẽ phụ thuộc vào content của TextView đó. TextView này lấy giá trị ban đầu ở trong thư mục res/values/string.xml. Đây là nơi định nghĩa các giá trị String.

Sau khi định nghĩa file XML, ta sẽ set layout này vào Activity ngay khi Activity được khởi động, tức là ở hàm onCreate(Bundle):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.main);
}
```

Như đã giới thiệu, ta có R chính là đối tượng R định nghĩa trong R.java. Khi ta thay đổi những tài nguyên của project, lập tức file R.java sẽ được cập nhật. Khi ta thay đổi file layout, thì R sẽ cập nhật sự thay đổi đó. Nên ta dễ dàng set layout cho Activity bằng hàm setContentView(R.layout.main), với main là tên của file XML định nghĩa layout cho Activity này.

Trong Android, ta có những kiểu Layout sau:

- LinearLayout
- RelativeLayout
- AbsoluteLayout
- TableLayout
- FrameLayout
- ScrollView

Thường thì trong chương trình ta sẽ kết hợp nhiều kiểu layout cho từng nhóm đối tượng tương tác để tạo ra layout mong muốn.

PHẦN II: PROJECT ANDROID THỰC HIỆN CHO LUẬN VĂN

I. Bluetooth

1. Giới thiệu

Bluetooth là phương thức dùng để truyền tín hiệu điều khiển từ điện thoại đến vi điều khiển STM32F4, vì vậy trong chương trình Android ta cần phải config các tính năng của Bluetooth. Do Android lập trình trên ngôn ngữ Java, một ngôn ngữ lập trình hướng đối tượng, để có thể tận dụng được tối đa những lợi thế của một ngôn ngữ lập trình hướng đối tượng, ta sẽ tạo một Class BluetoothConfig.java, với mục đích chính là mô tả một đối tượng BluetoothConfig chung, và ở các Activity ta chỉ cần gọi Instance của nó mà không cần phải lặp đi lặp lại các bước Config.

Nền tảng Android hỗ trợ chuẩn truyền không dây Bluetooth. Các tính năng của Bluetooth được sử dụng trong các ứng dụng có thể được lập trình thông qua các Bluetooth APIs. Những API này cho phép các thiết bị kết nối Bluetooth và các tính năng truyền nhận không dây.

Sử dụng Bluetooth API, ứng dụng Android có thể:

- Tìm kiếm các thiết bị Bluetooth.
- Truy vấn các thiết bị Bluetooth đã bắt cặp (paired devices).
- Thành lập các kênh RFCOMM.
- Kết nối tới các thiết bị khác thông qua service discovery.
- Truyền nhận dữ liệu tới và từ các thiết bị khác.

Để có thể sử dụng những tính năng chính của Bluetooth như: setting up, tìm kiếm thiết bị, kết nối thiết bị và truyền nhận dữ liệu giữa các thiết bị, ta sẽ sử dụng các Class được định nghĩa sẵn trong package android.bluetooth. Trong phạm vi luận văn ta sẽ sử dụng các Class sau:

- BluetoothAdapter: Class này đại diện cho Bluetooth Adapter của thiết bị điện thoại chạy hệ điều hành Android. Sử dụng Class này ta có thể tìm kiếm những thiết bị khác, truy vấn những thiết bị đã bắt cặp sẵn, hoặc tạo một Instance BluetoothDevice với tham số là địa chỉ MAC của thiết bị đó, và tạo một BluetoothServerSocket để nhận dữ liệu từ thiết bị khác.
- BluetoothDevice: đại diện cho các thiết bị trong phạm vi. Sử dụng Class này để yêu cầu kết nối với thiết bị đó thông qua BluetoothSocket hoặc truy vấn thông tin của thiết bị như: tên, địa chỉ, trạng thái, …

- BluetoothSocket: đại diện một giao diện (Interface) cho một Bluetooth Socket (tương tự như một TCP Socket). Đây là điểm kết nối cho phép ứng dụng truyền và nhận dữ liệu với thiết bị khác thông qua InputStream và OutputStream.

Để có thể sử dụng những tính năng của Bluetooth ở trong ứng dụng, ta phải khai báo quyền cho phép Bluetooth. Ta cần quyền này để khởi tạo giao tiếp Bluetooth.

Và như đã đề cập đến ở phần trước, để khai báo các quyền cho phép, ta cần khai báo chúng trong file AndroidManifest.xml.

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

2. Setting up Bluetooth

Trước khi ứng dụng có thể giao tiếp thông qua Bluetooth, ta cần kiểm tra xem thiết bị hiện tại có hỗ trợ Bluetooth hay không. Trong phạm vi luận văn, do thiết bị di động được sử dụng là điện thoại Nexus 4, có hỗ trợ Bluetooth, nên không cần phải kiểm tra. Nhưng nhìn chung khi ứng dụng được phát hành phổ biến thì không nên bỏ qua bước này.

Nếu thiết bị không được hỗ trợ Bluetooth, ta đành phải vô hiệu hóa toàn bộ các tính năng Bluetooth và phải nghĩ đến các phương án thay thế khác, còn nếu thiết bị có hỗ trợ, nhưng đang trong trạng thái "disabled", ta có thể chuyển sang trạng thái "enabled" mà không cần thoát khỏi ứng dụng. Ta có thể làm được điều này thông qua BluetoothAdapter.

- **Tạo một object BluetoothAdapter**

Để tạo được một object BluetoothAdapter, ta gọi hàm static getDefaultAdapter() của lớp BluetoothAdapter, hàm trả về một đối tượng BluetoothAdapter, đại diện cho Bluetooth Adapter mặc định của thiết bị (hay còn gọi là Bluetooth Radio). Thường chỉ có một Adapter cho một thiết bị, và ứng dụng có thể tương tác thông qua object của nó. Nêu hàm getDefaultAdapter() trả về null, nghĩa là thiết bị không hỗ trợ Bluetooth, và không thể thực hiện các tính năng của ứng dụng ta đang phát triển.

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth
}
```

- **Enable Bluetooth**

Tiếp theo, ta phải đảm bảo rằng Bluetooth đang được enable. Ta có thể gọi hàm isEnabled() để kiểm tra, nếu hàm trả về false, nghĩa là hiện tại đang disabled. Để có thể enable Bluetooth, ta sẽ bắt đầu một Activity của hệ thống, đó là Activity hỏi người dùng có muốn enable Bluetooth hay không. Và để bắt đầu một Activity mới, ta sẽ dùng Intent như đã giới thiệu ở phần trước.

```
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

Sẽ có một hộp thoại hiện lên hỏi người dùng có muốn bật Bluetooth hay không, nếu ta chọn YES, thì hệ thống sẽ bật Bluetooth và trở lại Activity cũ của ta.

Tuy nhiên, trong phạm vi của luận văn, vì ta muốn viết một Class để định nghĩa riêng cho các Bluetooth Config, nên nó sẽ không nằm trong Activity chính, vì vậy ta sẽ viết một hàm để kiểm tra trạng thái của Bluetooth, và bật nếu đang ở trạng thái disabled.

```
private final Handler mHandler;

…

	public void checkBtState() {
		if (btAdapter == null) {
			mHandler.sendEmptyMessage(BL_NOT_AVAILABLE);
		} else {
			if (btAdapter.isEnabled()) {
				Log.d(TAG, "Bluetooth On");
			} else {
				mHandler.sendEmptyMessage(BL_REQUEST_ENABLE);
			}
		}
	}
```

Ở đây ta không sử dụng Intent ngay, vì ta chỉ đang định nghĩa một Class BluetoothConfig, mà ta sẽ dùng một object của Class Handler, Class này giúp giải quyết các message gửi đến đợi hệ thống xử lý. Và ở các Activity sử dụng một đối tượng của BluetoothConfig ta sẽ có hàm để xử lý các message này. Và với message BT_REQUEST_ENABLE, ở hàm xử lý message (hàm handleMessage(Massage) mà ta sẽ giới thiệu sau) sẽ dùng một Intent để gửi yêu cầu bật Bluetooth.

3. Tìm kiếm thiết bị

Vì mạch vi điều khiển STM32F4 đã kết nối sẵn với một module Bluetooth, và module này có một địa chỉ MAC cố định, nên ta chỉ cần tìm đúng module này mà không cần phải bật tính năng tìm kiếm.

```
BluetoothDevice device = btAdapter.getRemoteDevice(address);
```

Trong đó hàm getRemoteDevice(String) lấy tham số là địa chỉ MAC của thiết bị, hàm trả về một object đại diện cho module Bluetooth, và ta có thể tiến hành các kết nối, truyền nhận dữ liệu thông qua object này.

4. Kết nối thiết bị

Để tạo được kết nối giữa hai thiết bị với nhau, ta phải lập trình cho cả Server và Client. Vì một thiết bị phải mở một Server Socket, và thiết bị còn lại phải khởi tạo kết nối (sử dụng địa chỉ MAC của server để khởi tạo kết nối). Server và Client được coi là đã kết nối với nhau nếu cả hai đều có chung một BluetoothSocket ở cùng kênh RFCOMM.

Tại thời điểm này, mỗi thiết bị có thể nhận các luồng input và output và có thể truyền dữ liệu.

Server và Client có được BluetoothSocket theo hai cách khác nhau. Server sẽ nhận được Socket khi nó chấp nhận kết nối được gửi tới. Client sẽ nhận được Socket khi nó mở một kênh RFCOMM đến server.

Một kỹ thuật lập trình được khuyên dùng là ở mỗi thiết bị, ta lập trình nó như một Server, nên mỗi thiết bị sẽ có một Socket luôn được mở và lắng nghe các kết nối gửi đến nó. Thiết bị còn lại có thể khởi tạo một kết nối với tư cách là Client.

Tuy nhiên, ở phạm vi luận văn, do ta đã biết địa chỉ MAC của module Bluetooth, mặt khác, module Bluetooth chỉ đóng vai trò truyền nhận tín hiệu, nên ta hoàn toàn có thể coi module Bluetooth là một Server, và thiết bị di động chạy hệ điều hành Android là một Client, khởi tạo kết nối đến nó.

- **Cách kết nối với vai trò một Client:**

Để có thể khởi tạo một kết nối với một thiết bị, đầu tiên ta phải có được một object BluetoothDevice đại diện cho thiết bị đó (đã trình bày ở trang trước). Sau đó ta sẽ dùng đối tượng này để tạo một BluetoothSocket và khởi tạo kết nối.

Quy trình có thể biểu diễn đơn giản như sau:

- Sử dụng object BluetoothDevice, nhận được một BluetoothSocket thông qua hàm createRfcommSocketToServiceRecord(UUID).

Hàm này sẽ khởi tạo một BluetoothSocket mà sẽ kết nối với thiết bị đại diện bởi BluetoothDevice object. Tham số UUID ở đây phải trùng với của Server khi nó mở BluetoothServerSocket của nó. Thông thường ta sẽ chọn UUID là chuỗi String:

```
private static final UUID MY_UUID = UUID
            .fromString("00001101-0000-1000-8000-00805F9B34FB");
```

- Khởi tạo kết nối bằng cách gọi hàm connect() của đối tượng BluetoothSocket vừa nhận được. Hàm connect() là một blocking-call (vì nếu vì lí do gì đó, kết nối bị lỗi hoặc quá 12 giây time out, hệ thống sẽ sinh ra lỗi), nên ta phải viết hàm này trong một luồng (Thread) tách biệt so với luồng của chương trình chính.

Hàm tạo Socket sử dụng trong luận văn:

```
	private BluetoothSocket createBluetoothSocket(BluetoothDevice device)
			throws IOException {
		if (Build.VERSION.SDK_INT >= 10) {
			try {
				final Method m = device.getClass().getMethod(

	"createInsecureRfcommSocketToServiceRecord",
					new Class[] { UUID.class });
				return (BluetoothSocket) m.invoke(device, MY_UUID);
			} catch (Exception e) {
				Log.e(TAG, "Could not create Insecure RFComm
Connection", e);
			}
		}
		return device.createRfcommSocketToServiceRecord(MY_UUID);
	}
```

Hàm trả về kiểu BluetoothSocket, tham số là đối tượng BluetoothDevice. Hàm sẽ kiểm tra phiên bản của Android hiện tại, nếu lớn hơn 10 (tương đương Android 2.3.3 trở lên), ta sẽ tạo Socket bằng hàm createInsecureRfcommSocketToServiceRecord(UUID), sử dụng hàm này vì ta chỉ sử dụng hàm createRfcommSocketToServiceRecord(UUID) trong trường hợp ta đảm bảo được độ bảo mật của kết nối, nếu một trong hai thiết bị không có khả năng nhận input và output, hoặc chỉ có khả năng hiển thị giá trị, thì không thể tạo ra một kết nối bảo mật, và Android từ 2.3.3 trở lên được khuyến cáo sử dụng hàm createInsecureRfcommSocketToServiceRecord(UUID).

Hàm tạo kết nối và tạo OutputStream:

```
	public boolean BT_Connect(String address, boolean listen_InStream) {
		Log.d(TAG, "...On Resume...");

		boolean connected = false;

		if (!BluetoothAdapter.checkBluetoothAddress(address)) {
			mHandler.sendEmptyMessage(BL_INCORRECT_ADDRESS);
			return false;
		} else {
			BluetoothDevice device = btAdapter.getRemoteDevice(address);
			try {
				btSocket = createBluetoothSocket(device);
			} catch (IOException e1) {
				Log.e(TAG,
						"In BT_Connect() socket create failed: "
								+ e1.getMessage());
				mHandler.sendEmptyMessage(BL_SOCKET_FAILED);
				return false;
			}
			btAdapter.cancelDiscovery();
			Log.d(TAG, "...Connecting...");
			try {
				btSocket.connect();
				Log.d(TAG, "...Connection OK...");
			} catch (IOException e) {
				try {
					btSocket.close();
				} catch (IOException e2) {
					Log.e(TAG,
							"In BT_Connect() unable to close
socket during connection failure"
									+ e2.getMessage());
					mHandler.sendEmptyMessage(BL_SOCKET_FAILED);
					return false;
				}
			}

			Log.d(TAG, "...Create socket...");
			try {
				outStream = btSocket.getOutputStream();
				connected = true;
			} catch (IOException e) {
				Log.e(TAG, "In BT_Connect() output stream creation
failed: "
						+ e.getMessage());
				mHandler.sendEmptyMessage(BL_SOCKET_FAILED);
				return false;
			}
		}
		return connected;
	}
```

Đầu tiên ta tạo một BluetoothSocket từ object BluetoothDevice được truyền vào từ tham số. Sau đó ta sẽ gọi hàm connect(). Nếu connect() bị lỗi thì ta sẽ gọi hàm close() để đóng nó lại.

Sau đó ta sẽ gọi hàm getOutputStream() để lấy OutputStream cho Socket. Đây là stream mà dữ liệu gửi đi từ thiết bị Android đi qua để đến module Bluetooth.

Thread độc lập để lắng nghe kết nối gửi đến thiết bị Android:

```
	private class ConnectedThread extends Thread {

		private final InputStream mInputStream;

		public ConnectedThread() {
			InputStream tmpIn = null;

			try {
				tmpIn = btSocket.getInputStream();
			} catch (IOException e) {
				Log.e(TAG,
						"In ConnectedThread() error
getInputStream(): "
							+ e.getMessage());
			}
			mInputStream = tmpIn;
		}

		public void run() {
			byte[] buffer = new byte[256];
			int bytes;

			while (true) {
				try {
					bytes = mInputStream.read(buffer);
				mHandler.obtainMessage(RECEIVE_MESSAGE, bytes, -1,
buffer)
						.sendToTarget();
				} catch (IOException e) {
					break;
				}
			}
		}
	}
```

Đầu tiên ta khởi tạo đối tượng của InputStream, đây là stream để dữ liệu đi từ bên ngoài vào thiết bị Android. Ta sẽ tạo bằng cách gọi hàm getInputStream() của đối tượng Socket đã có. Ở hàm run() của Thread này (đây là hàm mô tả các thao tác của Thread), ta đặt một biến buffer là một array kiểu byte với độ dài 256, và đọc về từ InputStream thông qua hàm read(buffer), giá trị đọc về sẽ được gửi đi bằng đối tượng của Class Handler như đã trình bày ở phần trước.

Trong phạm vi của luận văn, ta chỉ truyền dữ liệu từ thiết bị Android đến module Bluetooth kết nối sẵn với vi điều khiển STM32F4, nên ta tạm thời không sử dụng Thread này, vì vậy trong hàm BT_connect(String address, boolean listen_Instream),

tham số boolean sau ta sẽ để là false mỗi khi gọi hàm, tức là ta sẽ không sử dụng InputStream.

Để gửi được dữ liệu qua BluetoothSocket, ta sẽ định nghĩa một hàm sendData(String message) với tham số là message muốn gửi:

```
	public void sendData(String message) {
		byte[] msgBuffer = message.getBytes();
		Log.i(TAG, "Send data: " + message);

		if (outStream != null) {
			try {
				outStream.write(msgBuffer);
			} catch (IOException e) {
				Log.e(TAG,
					"In onResume() and exception occurred
during write: "
						+ e.getMessage());
				mHandler.sendEmptyMessage(BL_SOCKET_FAILED);
				return;
			}
		} else
			Log.d(TAG, "Error send data: outStream is NULL");
	}
```

Do ta phải gửi từng byte theo OutputStream, ta sẽ phải tạo một array msgBuffer kiểu byte để chứa chuỗi byte của message, sau đó gửi dữ liệu thông qua hàm write(byte) của đối tượng OutputStream.

Cuối cùng ta sẽ lập trình thêm một hàm nhỏ nữa, đó là hàm sử dụng khi ta muốn ngưng sử dụng tính năng Bluetooth, để tiết kiệm năng lượng cũng như bộ nhớ, ta sẽ đóng Socket và xóa OutputStream:

```
	public void BT_onPause() {
		Log.d(TAG, "...On Pause...");
		if (outStream != null) {
			try {
				outStream.flush();
			} catch (IOException e) {
				Log.e(TAG, "In onPause() and failed to flush output
stream: "
					+ e.getMessage());
				mHandler.sendEmptyMessage(BL_SOCKET_FAILED);
				return;
			}
		}

		if (btSocket != null) {
			try {
				btSocket.close();
			} catch (IOException e2) {
				Log.e(TAG,
					"In onPause() and failed to close socket:
"
					+ e2.getMessage());
				mHandler.sendEmptyMessage(BL_SOCKET_FAILED);
				return;
			}
		}
	}
```

Như vậy ta đã khảo sát xong những tính năng chính của Bluetooth trên hệ điều hành Android, cũng như cách tạo đối tượng BluetoothAdapter, BluetoothDevice, BluetoothSocket, các khởi tạo kết nối cũng như truyền nhận dữ liệu thông qua Bluetooth. Ở phần tiếp theo, ta sẽ khảo sát một Activity dùng để tạo một giao diện điều khiển ở chế độ Manual Mode.

II. Manual Mode Activity

Ở phần này, ta sẽ khảo sát một chế độ hoạt động của ứng dụng, đó là chế độ điều khiển bằng tay (Manual Mode). Và điều này đòi hỏi sự tương tác của người dùng trên thiết bị Android (chạm tay vào các Button để điều khiển xe), nên ta sẽ tạo một Activity để mô tả nó. Ta tạo file ManualMode.java để lập trình cho Activity này.

1. Layout

Ta sẽ tạo một layout cho Activity này và đặt tên là manual_mode_activity.xml. Để điều khiển xe, ta sẽ cần các 4 Button đảm nhiệm việc truyền các tín hiệu cơ bản:

- Forward: Dùng để ra lệnh đi tới
- Backward: Dùng để ra lệnh đi lùi
- Left: Dùng để ra lệnh rẽ trái
- Right: Dùng để ra lệnh rẽ phải

Ta sẽ định nghĩa bố cục của Layout là kiểu RelativeLayout, lợi điểm của RelativeLayout là nó định nghĩa vị trí của các đối tượng View một cách tương đối, ví dụ đối tượng A có thuộc tính "nằm bên phải" của đối tượng B, dẫn đến khi ứng dụng chạy trên những thiết bị có kích thước màn hình khác nhau thì Layout sẽ vẫn hiện thị đúng vị trí tương đối của các đối tượng.

Vậy ta có tổng cộng bốn Button, tương ứng cho bốn tính năng Forward, Backward, Left, Right. Để có thể canh đều thì ta sẽ định nghĩa thêm một Button nữa, gọi là Center Button, nhưng cho nó ẩn đi, vậy thì các nút tương ứng sẽ có một vị trí tương đối đối với Center Button này, tương ứng là trên, dưới, trái, phải.

Ví dụ với Forward Button:

```
<Button
    android:id="@+id/centerButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:visibility="invisible" />

<Button
    android:id="@+id/forward"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/centerButton"
    android:layout_centerHorizontal="true"
    android:text="@string/forward" />
```

Ở ví dụ này, ta thấy Forward Button có một thuộc tính là "layout_above" và được gán vào ID của Center Button, cho biết là nút Forward Button sẽ nằm ở trên Center Button.

- Giới thiệu về thuộc tính ID của các đối tượng View

Như đã đề cập ở phần giới thiệu giao diện của Activity, giao diện được tạo thành bởi các đối tượng thuộc các Class như Button, TextView, EditText, … Các Class này là các Subclass của Class View. Các đối tượng muốn được sử dụng trong các file .java thì ta cần một tham chiếu đến nó. Như đã giới thiệu, những thay đổi trong các folder hoặc file sẽ được build trong file R.java, và trong file R.java này có một nơi lưu trữ các ID cho các đối tượng View tạo nên giao diện Layout. Vì vậy trong các file source code (file .java) ta có thể dễ dàng gọi các đối tượng này thông qua "id" của chúng, ta sẽ xem xét ví dụ ở trên, ta đặt ID của Forward Button là "forward". Vì vậy trong file Activity ta có thể trỏ đến đối tượng này như sau:

```
    private Button forwardButton, backwardButton, leftButton, rightButton;

        forwardButton = (Button) findViewById(R.id.forward);
        backwardButton = (Button) findViewById(R.id.backward);
        leftButton = (Button) findViewById(R.id.left);
        rightButton = (Button) findViewById(R.id.right);
```

Đầu tiên ta định nghĩa các đối tượng Button dùng trong chương trình, sau đó ta sẽ trỏ đến các Button định nghĩa trong Layout thông qua hàm findViewById(int id).

2. ManualMode.java

Sau khi đã tạo xong Layout cho Activity, ta sẽ lập trình cho Activity này thông qua file ManualMode.java.

Đầu tiên ta sẽ tạo một đối tượng của Class BluetoothConfig, đối tượng này sẽ thực hiện các hàm yêu cầu các tính năng Bluetooth: như kiểm tra trạng thái, kết nối thiết bị và truyền dữ liệu:

```
    BluetoothConfig bt = null;
```

Sau đó tạo các đối tượng Button tương ứng với các Button định nghĩa trong Layout:

```
private Button forwardButton, backwardButton, leftButton, rightButton;
```

Như đã đề cập ở phần Life Cycle của một Activity, một Activity khi bắt đầu sẽ gọi hàm onCreate(Bundle), và ta sẽ set Layout cho Activity này thông qua hàm setContentView(int).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.manual_mode_activity);
```

Tiếp đến ta gán các biến Button tương ứng với các Button định nghĩa trong Layout:

```
	forwardButton = (Button) findViewById(R.id.forward);
	backwardButton = (Button) findViewById(R.id.backward);
	leftButton = (Button) findViewById(R.id.left);
	rightButton = (Button) findViewById(R.id.right);
```

Sau đó, ta sẽ khởi tạo đối tượng BluetoothConfig và kiểm tra trạng thái của Bluetooth:

```
	bt = new BluetoothConfig(this, mHandler);
	bt.checkBtState();
```

Ta khởi tạo đối tượng BluetoothConfig bằng Constructor định nghĩa trong file BluetoothConfig.java, lấy hai tham số là Context và Handler, ngụ ý là đối tượng BluetoothConfig sẽ thuộc về Activity này và đối tượng Handler của Activity này sẽ xử lý các message được gửi đi.

Hàm checkBtState() sẽ kiểm tra trạng thái Bluetooth, nếu đang disabled thì nó sẽ khởi động Activity của hệ thống để hỏi người dùng có cho phép bật Bluetooth hay không.

- **Hàm xử lý khi có tương tác của người dùng đến các Button**

Ở ngôn ngữ lập trình Java, chúng ta có khái niệm Listener, là các chương trình dùng để "lắng nghe" các sự kiện xảy ra trong quá trình tương tác của người dùng hoặc có sự thay đổi trong hệ thống, ví dụ như sự kiện Click chuột, Rê chuột, Gõ bàn phím, …

Và Android được viết trên ngôn ngữ Java, nên ta cũng sẽ có những "Listener" để nhận biết các tương tác của người dùng đến các đối tượng thông qua các sự kiện như Chạm màn hình, Click vào nút, …

Trong trường hợp Button, ta sẽ gọi hàm setOnTouchListener(OnTouchListener) để xử lý sự kiện khi người dùng chạm vào Button đó.

Ta cùng xem hàm xử lý cho Forward Button dưới đây:

```
forwardButton.setOnTouchListener(new OnTouchListener() {
        public boolean onTouch(View v, MotionEvent event) {
                if (event.getAction() == MotionEvent.ACTION_MOVE) {
                        if (btIsConnected) {
                                bt.sendData(String.valueOf("L+255R+255"));
                        }
                } else if (event.getAction() == MotionEvent.ACTION_UP) {
                        if (btIsConnected) {
                                bt.sendData(String.valueOf("S+000" + "S+000"));
                        }
                }
                return false;
        }
});
```

Hàm setOnTouchListener nhận tham số là một đối tượng OnTouchListener, và ta phải định nghĩa hàm onTouch(View, MotionEvent), trong đó View là để chỉ đối tượng View (dùng chung cho các đối tượng mà người dùng tương tác, và MotionEvent là các sự kiện chuyển động)

Khi MotionEvent là MOVE, tương ứng khi tay ta chạm và rê màn hình, thì ta sẽ sử dụng hàm sendData(String) để gửi đi tín hiệu điều khiển đi tới, và khi MotionEvent là UP, tương ứng khi ta nhấc ngón tay lên, thì ta sẽ gửi tín hiệu để dừng xe lại.

- **Giới thiệu về các lệnh điều khiển:**

Ta sẽ gửi đi tổng cộng 10 byte tới vi điều khiển. Trong đó 5 byte đầu dùng để điều khiển bánh xe bên trái và 5 byte sau dùng để điều khiển bánh xe bên phải.

Với mỗi 5 byte, byte đầu tiên là ký tự L hoặc R, tương ứng là trái hoặc phải, để vi điều khiển có thể xác nhận lại tín hiệu điều khiển, byte tiếp theo là "+" hoặc "-" tương ứng để điều khiển quay thuận hoặc quay nghịch, trong trường hợp này "+" tương ứng là chiều làm cho xe di chuyển về phía trước và ngược lại. 3 byte tiếp theo là giá trị PWM để điều khiển động cơ.

Vậy khi điều khiển ta sẽ có các trường hợp sau:

- L+xxx R+xxx: điều khiển xe đi về phía trước, tương ứng cả hai động cơ quay thuận.
- L-xxx R+xxx: điều khiển xe rẽ trái, tương ứng động cơ trái quay ngược và động cơ phải quay thuận.
- L+xxx R-xxx: điều khiển xe rẽ phải, tương ứng động cơ trái quay thuận và động cơ phải quay ngược.
- L-xxx R-xxx: điều khiển xe đi lùi, tương ứng cả hai động cơ quay ngược.

Tiếp theo ta sẽ cùng xét Class Handler dùng trong Activity này, vì ta muốn xử lý các message theo các trường hợp tương ứng, nên ta sẽ viết lại một Class MyHandler mới mà kế thừa Class Handler.

```
	private static class MyHandler extends Handler {
		private final WeakReference<ManualMode> mActivity;

		public MyHandler(ManualMode activity) {
			mActivity = new WeakReference<ManualMode>(activity);
		}
```

Ở đây ta sử dụng phương thức tham chiếu đến Activity của ta theo kiểu WeakReference, đây là kiểu tham chiếu thường được dùng khi ta muốn nó được loại bỏ tự động khi ta không tham chiếu đến nó nữa, giúp tiết kiệm được năng lượng và bộ nhớ.

Và ta đang thừa kế Class Handler, ta sẽ viết lại một hàm của nó (viết lại một hàm của SuperClass được gọi là Override), đó là hàm handleMessage(Message).

```
@Override
public void handleMessage(Message msg) {
	ManualMode activity = mActivity.get();
	if (activity != null) {
		switch (msg.what) {
		case BluetoothConfig.BL_NOT_AVAILABLE:
			Log.d(BluetoothConfig.TAG,
					"Bluetooth is not available. Exit");
			Toast.makeText(activity.getBaseContext(),
					"Bluetooth is not available", Toast.LENGTH_SHORT)
				.show();
			activity.finish();
			break;
		case BluetoothConfig.BL_INCORRECT_ADDRESS:
			Log.d(BluetoothConfig.TAG, "Incorrect MAC address");
			Toast.makeText(activity.getBaseContext(),
				"Incorrect Bluetooth address", Toast.LENGTH_SHORT)
				.show();
			break;
		case BluetoothConfig.BL_REQUEST_ENABLE:
			Log.d(BluetoothConfig.TAG, "Request bluetooth enabled");
			BluetoothAdapter.getDefaultAdapter();
			Intent enableBtIntent = new Intent(
					BluetoothAdapter.ACTION_REQUEST_ENABLE);
			activity.startActivityForResult(enableBtIntent, 1);
			break;
		case BluetoothConfig.BL_SOCKET_FAILED:
			Toast.makeText(activity.getBaseContext(), "Socket failed",
	Toast.LENGTH_SHORT).show();
			break;
		}
	}
}
```

Các message mà Class BluetoothConfig gửi đi gồm có:

- BT_NOT_AVAILABLE: message được gửi khi thiết bị không hỗ trợ Android. Khi này hệ thống sẽ gửi một dòng thông báo thông qua Class Toast, cho người dùng biết là Bluetooth không được hỗ trợ, sau đó sẽ ngưng Activity thông qua hàm finish()
- BT_INCORRECT_ADDRESS: message được gửi khi không thể kết nối với thiết bị do sai địa chỉ.
- BT_REQUEST_ENABLE: message được gửi khi ta yêu cầu bật Bluetooth, và hệ thống sẽ bắt đầu Acvtivity của hệ thống thông qua Intent.
- BT_SOCKET_FAILED: message được gửi khi Socket không thể tạo được.

Tương ứng với các message mà hàm handleMessage(Message) sẽ có phương thức xử lý giống như đã định nghĩa ở phía trên.

Cuối cùng, ta còn hai hàm nữa để mô tả hành vi cho Activity trong quá trình chuyển trạng thái trong Life Cycle, đó là onPause() và onResume(). Ở hàm onPause() ta chỉ đơn giản là gọi hàm BT_onPause() của đối tượng BluetoothConfig, và kèm thêm điều kiện là Bluetooth đang được enabled, nếu không sẽ phát sinh lỗi, do hàm BT_onPause() sẽ xóa OutputStream, mà khi đó OutputStream chưa phải là một đối tượng.

Còn trong hàm onResume(), ta cũng sẽ gọi hàm BT_Connect(String, boolean) để kết nối lại với thiết bị.

```
	@Override
	protected void onResume() {
		super.onResume();
		btIsConnected = bt.BT_Connect(address, false);
	}

	@Override
	protected void onPause() {
		super.onPause();
		if (BluetoothConfig.btAdapter.isEnabled()) {
			bt.BT_onPause();
		}
	}
```

Ta đã khảo sát xong chế độ Manual được mô tả trong file ManualMode.java. Ở chế độ này ứng dụng hoạt động và điều khiển đúng tín hiệu. Ở bên đáp ứng của vi điều khiển cũng đọc đúng tín hiệu điều khiển và hoạt động tốt.

III. Auto Mode

Sau khi khảo sát xong chế độ điều khiển bằng tay (Manual Mode), tiếp theo ta sẽ khảo sát đến chế độ tự động (Auto Mode).

Vì đây là ứng dụng hướng đến mục đích là điều khiển một chiếc xe với các tính năng như:

- Nhận biết được vị trí hiện tại.
- Tìm được đường đi từ vị trí hiện tại đến vị trí muốn đến.
- Tự tính toán các góc quay giữa các điểm dừng.
- Tính toán khoảng cách và đưa ra được tín hiệu điều khiển phù hợp.

Với các tính năng mong muốn như trên, trong phạm vi của luận văn ta sẽ cần nghiên cứu sử dụng đến các hệ thống và phần cứng như sau:

- GPS
- Google Maps
- Các cảm biến của thiết bị Android

Đầu tiên ta sẽ khảo sát tổng quát những gì ta mong muốn ở những mục trên.

- GPS: GPS – Global Positioning System, là hệ thống định vị toàn cầu của Mỹ, và được sử dụng rộng rãi trên toàn thế giới. Ta sẽ sử dụng GPS để định vị được vị trí hiện tại. Ở phần sau ta sẽ khảo sát kỹ hơn về tính năng của GPS và cách thức lập trình để sử dụng GPS trên thiết bị Android.
- Google Maps: đây là một ứng dụng nổi tiếng của Google. Ta sẽ nhúng bản đồ vào Layout của Activity, và hiển thị vị trí hiện tại của ta lên bản đồ, sau đó chọn địa điểm muốn đến bằng cách chạm vào một điểm khác mà ta mong muốn. Và dựa vào Google Maps API ta sẽ lập trình để có được tập hợp các tọa độ mà ta cần đi qua, và tính toán được các tín hiệu điều khiển.
- Các cảm biến của thiết bị Android: từ các tập hợp tọa độ có được từ Google Maps API, ta sẽ tính được các góc quay cần thiết, tuy nhiên ta cần tính được góc quay hiện tại của thiết bị Android để đối chiếu với góc cần quay. Và ta sẽ thực hiện điều đó bằng cách đọc các giá trị của các cảm biến trong thiết bị Android. Các cảm biến ta sẽ sử dụng là Accelerometer và Magnetic Field. Từ giá trị của các cảm biến, ta sẽ tính toán được góc quay hiện tại của thiết bị Android.

Tiếp theo ta sẽ khảo sát chi tiết từng module ở trên.

1. GPS – Location Based Service

Một trong những tính năng mà thiết bị di động hiện tại đều phải tối ưu, đó chính là tính gọn nhẹ và tiện lợi. Người sử dụng cần một thiết bị cầm tay mà được tích hợp những thứ cần thiết, một trong số đó là khả năng định vị vị trí. Và trong thời đại công nghệ thông tin như hiện nay, có rất nhiều API hỗ trợ chúng ta trong việc tìm kiếm, hiển thị và đánh dấu các địa điểm.

Trong phạm vi luận văn, ta sẽ sử dụng Location Based Service – LBS để xác định vị trí hiện tại của thiết bị Android. Có nhiều kỹ thuật để thực hiện được điều này, và ta có thể chọn sử dụng một kỹ thuật nhất định, hoặc đưa ra những yêu cầu (về độ chính xác, tiết kiệm năng lượng, …) để hệ thống sẽ chọn cho chúng ta kỹ thuật nào phù hợp nhất.

- **Sử dụng Location Based Service**

Location Based Service là một từ dùng để chỉ các kỹ thuật khác nhau mà ta có thể sử dụng để xác định vị trí hiện tại. Có hai phần tử chính của LBS là:

- Location Manager: tạo đối tượng sử dụng LBS.
- Location Providers: bao gồm các kỹ thuật xác định vị trí khác nhau mà ta sử dụng để xác định vị trí hiện tại.

Sử dụng Location Manager, ta có thể:

- Tìm được vị trí hiện tại.
- Theo dõi sự di chuyển.
- Tạo được các cảnh báo tiệm cận (proximity alert) khi phát hiện chuyển động đi vào hoặc đi ra khỏi một khu vực chỉ định.
- Tìm những Location Providers được hỗ trợ.
- Theo dõi trạng thái của bộ thu GPS.

Ta có thể có được sự truy cập đến LBS thông qua Location Manager. Để truy cập được Location Manager, ta sẽ yêu cầu khởi tạo một Instance của LOCATION_SERVICE sử dụng hàm getSystemService:

```
LocationManager locationManager;

svcName = Context.LOCATION_SERVICE;

locationManager = (LocationManager) getSystemService(svcName);
```

Trước khi ta có thể sử dụng LBS, ta phải khai báo các quyền cho phép vào file AndroidManifest.xml, giống như đã khai báo quyền truy cập Bluetooth ở phần trước.

Ta sẽ khai báo quyền cho phép đối với “Fine Location” và “Coarse Location” như sau:

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

Fine và Coarse là hai khái niệm thể hiện mức độ chính xác mà ta có thể có được khi định vị vị trí, trong đó Fine có độ chính xác cao hơn và Coarse có độ chính xác thấp hơn.

- **Lựa chọn Location Providers:**

Tùy thuộc vào thiết bị Android của mỗi người, ta có thể sử dụng những kỹ thuật khác nhau để xác định vị trí. Mỗi kỹ thuật có những khả năng khác nhau: bao gồm sự khác nhau về độ tiêu hao năng lượng, độ chính xác, và khả năng xác định thông tin về độ cao, tốc độ, …

Class LocationProvider bao gồm ba hằng số, tưng ứng trả về tên của provider của ba Location Providers:

- LocationManager.GPS_PROVIDER
- LocationManager.NETWORK_PROVIDER
- LocationManager.PASSIVE_PROVIDER

GPS Provider và Passive Provider yêu cầu quyền cho phép “Fine”, trong đó Network Provider chỉ yêu cầu quyền cho phép “Coarse”.

- **Lựa chọn Provider dựa vào các điều kiện mong muốn**

Trong đa số các trường hợp, ta thường không thể chọn được đúng Provider mà mình muốn sử dụng. Khi đó, ta nên chỉ ra những yêu cầu mình mong muốn và để Android lựa chọn Provider giúp chúng ta.

Để làm được điều này, ta sẽ dùng Class Criteria để chứa các yêu cầu theo độ chính xác, độ tiêu hao năng lượng, tiêu hao tài chính, và khả năng xác định được độ cao, tốc độ, …

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_COARSE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setSpeedRequired(false);
criteria.setCostAllowed(true);
```

Sau khi định nghĩa đối tượng Criteria, ta có thể sử dụng hàm getBestProvider để trả về đối tượng Location Provider thích hợp nhất.

```
Criteria criteria = new Criteria();
String bestProvider = locationManager.getBestProvider(criteria, true);
```

Nếu có nhiều hơn một Provider thỏa mãn Criteria, thì Provider với độ chính xác cao hơn sẽ được chọn. Nếu không có Provider nào thỏa mãn, thì đối tượng Criteria sẽ giảm dần các điều kiện theo thứ tự sau, cho đến khi tìm được Provider thỏa mãn:

- Độ tiêu hao năng lượng
- Độ chính xác của vị trí trả về
- Độ chính xác của góc phương vị, tốc độ, độ cao
- Khả năng đọc được góc phương vị, tốc độ, độ cao

Điều kiện về sự tiêu tốn cước di động không bao giờ được giảm, nếu không có Provider nào phù hợp, hàm sẽ trả về null.

Để có được một Instance của một Provider mà ta muốn đích danh, ta cần gọi hàm getProvider(String) với tham số là tên của Provider đó.

```
String provider = LocationManager.NETWORK_PROVIDER;
LocationProvider gpsProvider = locationManager.getProvider(provider);
```

Ở phạm vi của luận văn, do phần cứng tương đối mang tính chất mô hình, và chưa thể so sánh với thực tế, nên để có thể dễ dàng trong quá trình demo, thay vì dùng GPS Provider thì ta sẽ dung Network Provider, vì GPS tuy độ chính xác cao nhưng cập nhật rất chậm do cơ chế hoạt động của GPS cần thu thập thông tin của ít nhất ba vệ tinh GPS thì mới trả kết quả vị trí về, và thực tế trong quá trình thực hiện luận văn, GPS Provider đáp ứng không như mong đợi với thời gian trễ giữa hai lần cập nhật là tương đối lâu, còn Network Provider tuy độ chính xác thấp hơn, nhưng tốc độ cập nhật rất nhanh.

- **Tìm vị trí cuối cùng được ghi nhận (Last Known Location)**

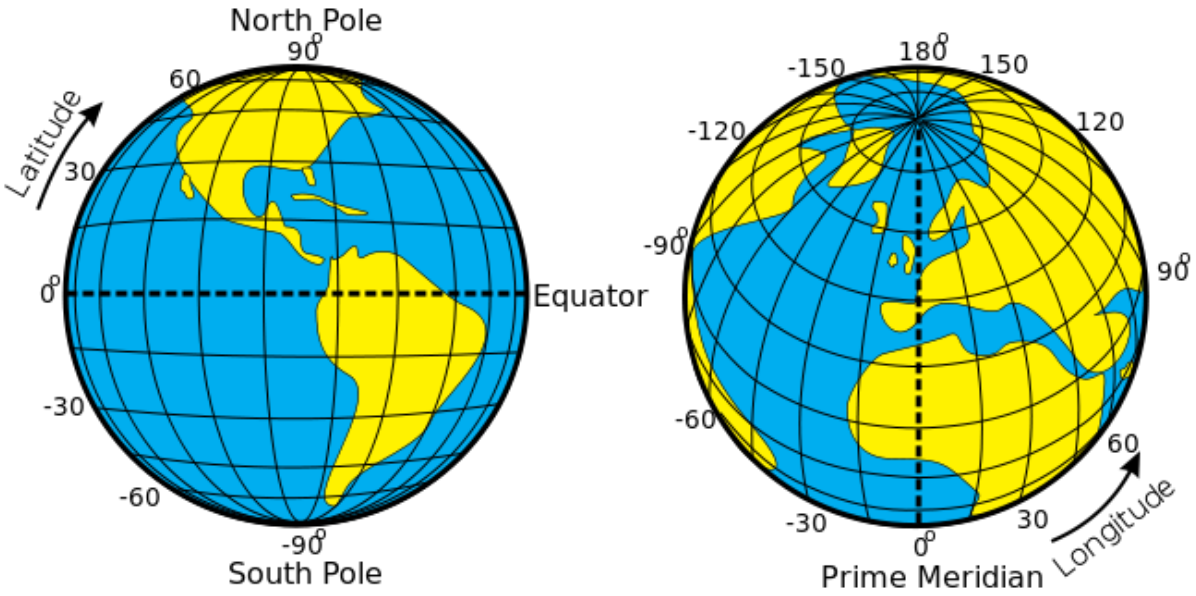
Ta có thể nhận được vị trí cuối cùng mà hệ thống ghi nhận, ta sẽ gọi hàm getLastKnownLocation, với tham số là tên của Provider được sử dụng. Ví dụ như ta muốn tìm vị trí cuối cùng được ghi nhận bằng Network Provider:

```
            provider = LocationManager.NETWORK_PROVIDER;
            l = locationManager.getLastKnownLocation(provider);
```

Lưu ý rằng hàm getLastKnownLocation không yêu cầu sự cập nhật mới từ Location Provider, nên vị trí nhận được có thể không còn chính xác nữa.

- **Giới thiệu về đối tượng Location, thuộc tính Latitude và Longitude**

Khi cập nhật vị trí, đối tượng Location sẽ được cập nhật, một đối tượng Location chứa bên trong rất nhiều thông tin về vị trí hiện tại, trong đó có hai thông tin quan trọng nhất, chính là Latitude và Longitude.



- Latitude: giá trị của vĩ độ, tính bằng độ, kiểu double.
- Longitude: giá trị của tung độ, tính bằng độ, kiểu double.

Với hai giá trị Latitude và Longitude mà ta thu được từ đối tượng Location, ta hoàn toàn có thể tính toán được khoảng cách giữa hai điểm, tính góc quay cần thiết, …

Sau khi đã nhận được đối tượng Location và lấy được vị trí cuối cùng mà hệ thống ghi nhận, ta sẽ tìm hiểu phương thức để cập nhật vị trí liên tục.

Đầu tiên ta sẽ thêm vào hàm onCreate() của Activity để tạo tham chiếu đến Class LocationManager, khởi tạo đối tượng Location bằng cách gán vị trí cuối cùng được hệ thống ghi nhận thông qua hàm getLastKnownLocation, và sau đó truyền vào một hàm updateWithNewLocation(Location) mà ta định nghĩa.

```
        svcName = Context.LOCATION_SERVICE;
        locationManager = (LocationManager) getSystemService(svcName);
        provider = LocationManager.NETWORK_PROVIDER;
        l = locationManager.getLastKnownLocation(provider);
        updateWithNewLocation(l);
```

Để có thể theo dõi được quá trình cập nhật, ta sẽ tạo trọng Layout một đối tượng TextView để quan sát sự thay đổi của hai thông tin đã giới thiệu ở trên:

```
    <TextView
        android:id="@+id/myLocationText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

Với ID đã biết của đối tượng TextView, ta sẽ tạo một TextView tương ứng trong Activity và tham chiếu đến đối tượng tương ứng trong Layout:

```
TextView myLocationText;

myLocationText = (TextView) findViewById(R.id.myLocationText);
```

Tiếp theo, ta sẽ định nghĩa hàm updateWithNewLocation(Location) mà ta đã gọi ở hàm onCreate(), mục đích của hàm updateWithNewLocation là sau khi nhận tham số là một đối tượng Location, ta sẽ lấy hai thông tin Latitude, Longitude để hiển thị lên Layout:

```
private final void updateWithNewLocation(Location location) {
        double lat = 0;
        double lng = 0;
        myLocationText = (TextView) findViewById(R.id.myLocationText);

        String latLngString = "No location found";
        if (location != null) {
                lat = location.getLatitude();
                lng = location.getLongitude();

        }

        latLngString = "Lat: " + lat + "\nLong: " + lng;
        myLocationText.setText("Your current position is:\n" + latLngString
                        + "\n" + controlSignal);

}
```

Ta lấy giá trị Latitude bằng cách gọi hàm getLatitude() và Longitude bằng cách gọi hàm getLongitude(), sau đó ta sẽ cho hai giá trị này vào một chuỗi String, cho hiển thị bằng hàm setText(String) của đối tượng TextView.

Ở trên, ta mới cho hiển thị thông tin của vị trí, nhưng lại chưa có phương thức để cập nhật vị trí để gửi vào hàm updateWithNewLocation(Location). Trong khi ứng dụng đang phát triển cần nhận biết được sự thay đổi của vị trí để có thể điều khiển. Để có thể cập nhật vị trí liên tục, ta sẽ gọi hàm requestLocationUpdates(provider, minTime, minDistance, listener). Trong đó provider chính là đối tượng Provider mà ta đang sử dụng, minTime là thời gian tối thiểu giữa hai lần cập nhật mà ta mong muốn, minDistance là khoảng cách tối thiểu để cập nhật, listener là phương thức lắng nghe khi có sự kiện vị trí thay đổi.

Và ta sẽ phải định nghĩa một Location Listener để dùng cho hàm requestLocationUpdates này.

```
private final LocationListener locationListener = new LocationListener() {
	public void onLocationChanged(Location location) {
		updateWithNewLocation(location);
	}

	public void onProviderDisabled(String provider) {
	}

	public void onProviderEnabled(String provider) {
	}

	public void onStatusChanged(String provider, int status, Bundle extras) {
	}
};
```

Ta định nghĩa một đối tượng LocationListener, và ta Override các hàm của một đối tượng LocationListener:

- onLocationChanged: hàm xử lý khi vị trí hiện tại thay đổi. Trong hàm này ta gọi hàm updateWithNewLocation(Location) để cập nhật thông tin Latitude và Longitude ở Layout.
- onProviderDisabled: hàm xử lý khi Provider chuyển sang trạng thái disabled.
- onProviderEnabled: hàm xử lý khi Provider chuyển sang trạng thái enabled.
- onStatusChanged: hàm xử lý khi trạng thái của Provider thay đổi, ví dụ như Provider không thể lấy được thông tin vị trí, …

Trong phạm vi luận văn ta chỉ sử dụng hàm xử lý khi vị trí hiện tại thay đổi onLocationChanged(Location).

Sau khi định nghĩa một đối tượng locationListener, ta sẽ thêm hàm requestLocationUpdates vào hàm onCreate() của Activity.

```
// Location Service
svcName = Context.LOCATION_SERVICE;
locationManager = (LocationManager) getSystemService(svcName);
provider = LocationManager.NETWORK_PROVIDER;
l = locationManager.getLastKnownLocation(provider);
dest = new LatLng(l.getLatitude(), l.getLongitude());
updateWithNewLocation(l);
locationManager.requestLocationUpdates(provider, 1000, 0,
            locationListener);
```

Trong đó, minTime ta để là 1000 ms, tương ứng thời gian tối thiểu giữa hai lần cập nhật là 1 giây, và ta để minDistance là 0m, tương ứng sẽ không yêu cầu khoảng cách tối thiểu cho cập nhật.

Như vậy, chúng ta đã khảo sát xong những đặc trưng của GPS - Location Based Service cũng như cách thức sử dụng các tính năng định vị và cập nhật vị trí trong chương trình Android. Tiếp theo ta sẽ khảo sát một trong những module quan trọng nhất trong giải thuật, chính là Google Maps API.

2. Google Maps API

Google Maps là hệ thống bản đồ toàn cầu nổi tiếng của Google. Thông qua các ứng dụng trên nền tảng PC và di động, chắc hẳn chúng ta đều đã được trải nghiệm những tính năng tuyệt vời của hệ thống bản đồ này trong việc xác định các địa điểm, dò lộ trình giữa hai địa điểm, …

Vậy làm thế nào để ta có thể viết riêng cho mình một ứng dụng có thể sử dụng các tính năng của Google Maps? Google Maps API chính là câu trả lời cho chúng ta.

Google Maps API là một API cung cấp nhiều hàm cho phép chúng ta xây dựng những ứng dụng mà có thể truy cập những tính năng của Google Maps, ví dụ như:

- Hiển thị bản đồ Google Maps trên Layout của Activity.
- Tạo các điểm đánh dấu trên bản đồ và lấy được các giá trị Latitude & Longitude tương ứng của chúng.
- Tìm thông tin tuyến đường.
- Tìm lộ trình giữa hai địa điểm.
- Vẽ các đường nối giữa các điểm nằm giữa hai địa điểm.

Google Maps API hỗ trợ ta rất nhiều trong việc xác định những thông tin của địa điểm, lộ trình, tuy nhiên việc sử dụng được Google Maps API cũng đòi hỏi một vài tùy chỉnh cấu hình và các thủ tục khác. Mặt khác, bản thân Google Maps API sẽ không hoàn toàn giúp ta hoàn thành các công việc trên, ta vẫn phải định nghĩa thêm nhiều hàm giải thuật khác nhau và cùng kết hợp thì mới có thể giải quyết được công việc nêu trên. Quy trình bao gồm:

- Tùy chỉnh để có thể sử dụng Google Maps API.
- Dựa vào API ta sẽ nhận được một file JSON chứa thông tin lộ trình của hai địa điểm mong muốn.
- Viết các Class và hàm để quy đổi file JSON về thành những chuỗi giá trị tương ứng với các địa điểm trung gian.
- Vẽ lộ trình đi qua các địa điểm nằm trong chuỗi ở trên.
- Cùng với Location Based Service và kết quả đọc được từ các Sensor, ta sẽ đưa ra các giải thuật điều khiển phù hợp.

Do là module quan trọng nhất, nên ta sẽ đi chi tiết vào từng phần nhỏ, và sẽ xét đến giải thuật điều khiển sau cùng. Bây giờ ta sẽ khảo sát các bước tùy chỉnh cần thiết để có thể sử dụng Google Maps API.

a. **Tùy chỉnh để sử dụng được Google Maps API**
   - Đầu tiên, ta cần có Android SDK, điều này là dĩ nhiên và ta đã hoàn thành ở phần I khi giới thiệu hệ điều hành Android.
   - Sau đó ta cần cài đặt Google Play Service SDK. Như đã nói ở phần trước, Google Play Service SDK chứa rất nhiều bộ API của Google, trong đó có Google Maps API, và ta cũng đã làm xong công đoạn này ở phần I.
   - Lấy API Key, đây là một Key cần thiết để ứng dụng của ta có thể truy cập được các tính năng của Google Maps.
   - Khai báo các quyền cho phép trong file AndroidManifest.xml.
   - Thêm bản đồ vào trong Activity của ứng dụng.

Sau khi đã cài đặt Android SDK và Google Play Service SDK, ta cần khai báo việc sử dụng Google Play Service SDK trong file AndroidManifest.xml, khai báo này đặt bên trong phần tử <application>

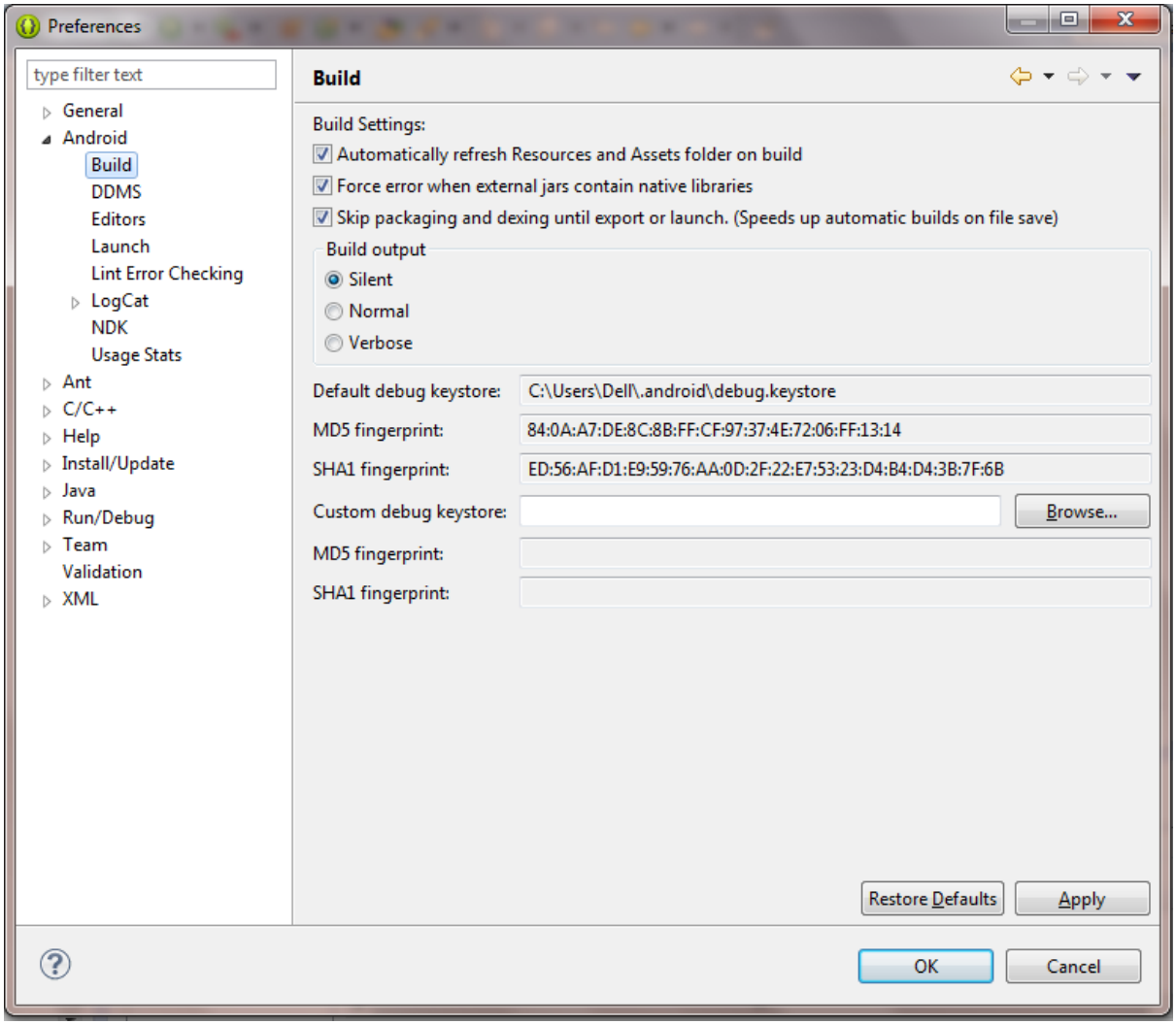
```
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
```

- **Lấy chứng chỉ Android (Android Certificate) và Google Maps API Key**

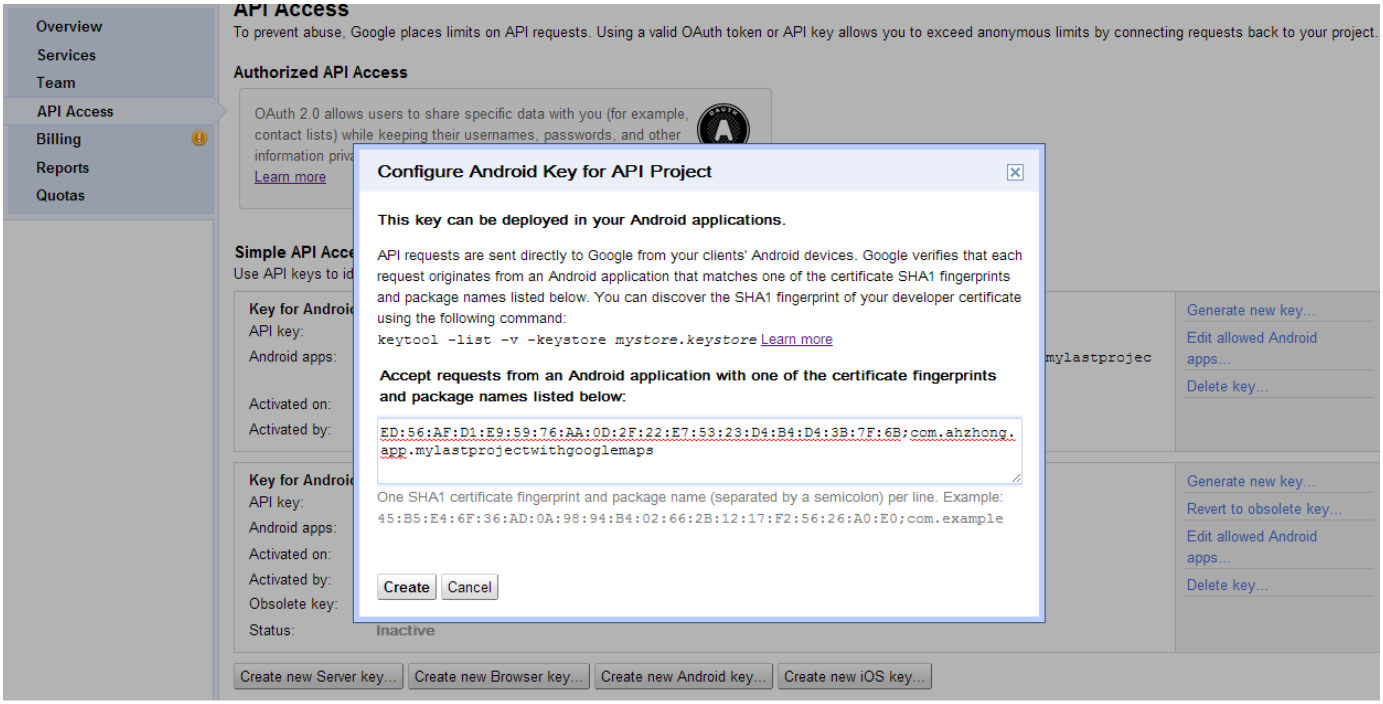
Phiên bản API mà ta đang sử dụng là phiên bản version 2, ta cần lưu ý điều này vì hiện tại Google không còn hỗ trợ version 1 nữa. Vì vậy Key của version 1 sẽ không thể sử dụng lại cho version 2.

Để có thể truy cập được Google Maps Server bằng Maps API, ta phải có khai báo API Key trong ứng dụng. Key này hoàn toàn miễn phí và có thể sử dụng cho bất kỳ ứng dụng nào muốn gọi Maps API và không giới hạn số lượng người dùng. Ta sẽ nhận API Key này từ Google API Console bằng cách cung cấp chứng chỉ Android và tên của package của ứng dụng.

Để có được chứng chỉ Android, trước đây ta phải chạy các lệnh trong command promt để hiển thị, tuy nhiên hiện tại, với IDE Eclipse ta dễ dàng có được thông tin này bằng cách: Vào Windows -> Preferences -> Android -> Build, tại đây ta có được thông tin của hai dạng mã hóa của chứng chỉ Android, bao gồm mã hóa MD5 và SHA-1. Thông tin mã hóa SHA-1 chính là chuỗi ta cần.



Sau khi lấy được mã hóa SHA-1 của chứng chỉ Android, ta sẽ truy cập vào Google API Console. Chọn API Access -> Create New Android Key. Ở cửa sổ hiện lên ta điền mã hóa SHA-1 ở trên, theo sau bởi dấu “;” và tiếp theo là tên package của project.



Sau khi lấy xong được API Key, những công đoạn config còn lại tương đối đơn giản, đó là khai báo các quyền cho phép trong file AndroidManifest.xml, chỉnh sửa file Layout của Activity và lập trình file mã nguồn của Activity để giao tiếp với Google Maps.

- **Khai báo API Key vào ứng dụng**

Ta sẽ khai báo API Key vào trong file AndroidManifest.xml. Sau đó, Maps API sẽ đọc API Key này và gửi đến Google Maps Server, sau đó sẽ xác nhận rằng ta có quyền truy cập dữ liệu Google Maps.

Ở file AndroidManifest.xml, ta thêm phần tử dưới đây bên trong phần tử <application>, bằng cách thêm vào ngay phía trên của tag </application>

```
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyCnY9e1_GlcC71tGTGA6S-u78bCEpj0bqk" />
    </application>
```

Trong đó ở thuộc tính value ta điền API Key mà ta đã nhận được ở các bước như phía trên.

Sau đó ta lưu lại file AndroidManifest.xml và build lại ứng dụng.

- **Khai báo các thiết lập trong file AndroidManifest.xml**

Sau khi đã hoàn thành các bước ở trên (tham chiếu đến Google Play Service, khai báo API Key cho ứng dụng), ta cần làm thêm những công đoạn sau:

- Khai báo các quyền cho phép (permissions) mà cho phép ứng dụng được phép truy cập các tính năng hệ thống của Android và Google Maps Server.
- Thông báo cho biết ứng dụng yêu cầu sử dụng OpenGL ES version 2. Các service bên ngoài có thể phát hiện điều này và hành xử tương ứng.

Khai báo các quyền cho phép theo cú pháp sau:

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

Ta cần khai báo các quyền sau:

- android.permission.INTERNET: sử dụng để download Maps từ Google Maps Server.
- android.permission.ACCESS_NETWORK_STATE: cho phép API kiểm tra trạng thái kết nối để quyết định có thể download dữ liệu hay không.
- com.google.android.providers.gsf.permission.READ_GSERVICES: cho phép API truy cập Google Web-based Service.
- android.permission.WRITE_EXTERNAL_STORAGE: cho phép API lưu tạm (cache) dữ liệu bản đồ ở bộ nhớ ngoài của thiết bị.

Sau khi khai báo các quyền cho phép, ta cần khai báo yêu cầu sử dụng OpenGL ES version 2 để render bản đồ. Nếu không có OpenGL ES v2, bản đồ sẽ không xuất hiện trong ứng dụng. Ta sẽ thêm phần tử sau bên trong phần tử <manifest>:

```
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

Sau khi hoàn thành các bước tùy chỉnh, ta sẽ tiến hành hiển thị bản đồ trên Activity. Để làm được điều này, ta sẽ cần tùy chỉnh Layout và file mã nguồn của Activity.

•