Ôn lại kiến thức lập trình Android

Mục Lục

1. Java Code Convention 4

1.1. Mục đích quy ước viết Code 4

1.2. Các quy tắc chung 4

1.3. Tổ chức file mã nguồn JAVA 4

1.4. Chú thích 5

1.5. Khai báo, câu lệnh 5

1.6. Quy ước đặt tên 5

1.6.1. Quy ước chung 5

1.6.2. Class/Interface 6

1.6.3. Biến số 6

1.6.4. Hằng số 6

1.6.5. Phương thức 7

2. Android 8

2.1. Activity 8

2.2. Fragment 10

2.3. Theme/Style 11

2.4. UI Layout 11

2.4.1. FrameLayout 11

2.4.2. LinearLayout 12

2.4.3. RelativeLayout 12

2.5. UI Controls 12

2.5.1. ListView 12

2.5.2. GridView 12

2.5.4. Toolbar 13

2.6. Service 14

2.7. Database 15

2.8. Thread 15

2.8.1. AsyncTask 15

2.8.2. Handler 16

2.8.3. Thread 16

Tài liệu tham khảo 17

Lời nói đầu

Trong báo cáo này sẽ có chỗ nhận xét qua kinh nghiệm bản thân trải qua vậy nên nó có thể đúng hay sai. Mong mọi người góp ý

**1. Java Code Convention** [[1](http://www.oracle.com/technetwork/java/codeconventions-150003.pdf)]

Lập trình ứng dụng Android sẽ sử dụng nhiều đến ngôn ngữ JAVA nên tôi sẽ đề cập đến các tiêu chuẩn của JAVA

**1.1. Mục đích quy ước viết Code**

* 80% chi phí cho vòng đời của một phần mềm là dành cho việc bảo trì.
* Hầu như bất cứ phần mềm nào cũng được bảo trì toàn bộ bởi người viết phần mềm đó.
* Quy ước về viết code cải thiện tính dễ đọc của phần mềm, cho phép lập trình viên có thể hiểu code mới một cách nhanh chóng và hoàn thiện.

**1.2. Các quy tắc chung**

* Code đơn giản, dễ hiểu (comment đầy đủ nữa)
* Code phải chính xác và chắc chắn
* Luôn luôn có tư tưởng tối ưu

**1.3. Tổ chức file mã nguồn JAVA**

Mỗi file mã nguồn Java chứa một public class đơn hoặc một interface.

Bảng mô tả các thành phần trong file

|  |  |  |
| --- | --- | --- |
| No | Phần khai báo của Class/Interface | Ghi chú |
| 1 | Chú thích ban đầu  /\*  \* Classname  \*  \* Version info  \*  \* Copyright notice  \*/ |  |
| 2 | package java.awt;  import java.awt.peer.CanvasPeer;  import ….. | Khai báo package phải trên import |
| 3 | Chú thích tài liệu Class/interface  (/\*\*...\*/) | Tạo tài liệu cho Class/interface |
| 4 | Câu lệnh class or interface |  |
| 5 | Chú thích cài đặt Class/interface (nếu cần thiết)  (/\*...\*/) | Phần chú thích này chứa các thông tin về class hoặc interface mà không thuộc vào phần chú thích tài liệu. |
| 6 | Chú thích tài liệu hằng số (/\*\*….\*/)  Các hằng số (static final) | Đầu tiên là các hằng số public, sau đó đến protected, tiếp theo là hằng số mức package (không có access modifier), và cuối cùng là private. |
| 7 | Chú thích tài liệu biến Class (/\*\*….\*/)  Các biến của Class (static) | Đầu tiên là các biến public, sau đó đến protected, tiếp theo là hằng số mức package (không có access modifier), và cuối cùng là private. |
| 8 | Chú thích tài liệu biến của instance (/\*\*….\*/)  Các biến của instance | Đầu tiên là các biến public, sau đó đến protected, tiếp theo là hằng số mức package (không có access modifier), và cuối cùng là private. |
| 9 | Chú thích tài liệu Constructors (/\*\*….\*/)  Constructors |  |
| 10 | Chú thích tài liệu các phương thức (/\*\*….\*/)  Các phương thức | Các phương thức này nên được nhóm theo chức năng, chứ không nên nhóm theo phạm vi hoặc khả năng truy nhập. Ví dụ, một phương thức của lớp có kiểu private có thể nằm giữa 2 phương thức của instance kiểu public. Mục đích là làm cho việc đọc hiểu code dễ dàng hơn. |
| 11 | Inner classes/interfaces | Khai báo classes/interfaces trong class trên |

**1.4. Chú thích**

* Implementation comments như /\*...\*/ và //
* Documentation comments được giới hạn trong /\*\*...\*/ và có thể xuất thành các file HTML khi dùng javadoc tool

**1.5. Khai báo, câu lệnh**

Mỗi 1 dòng chỉ chứa 1 câu lệnh hoặc chỉ chứa 1 khai b

**1.6. Quy ước đặt tên** (Naming Conventions)

**1.6.1. Quy ước chung**

1. Chữ cái đầu tiên phải là kí tự chữ cái
2. Người lập trình khi đặt tên phải **có ý nghĩa**, và phải nó phải chỉ ra được mục đích của file / variable / control / method.
3. Sử dụng **terminology applicable to the domain**.   
   Nếu người dùng của bạn coi các clients của họ như là các customers, khi đó ta phải sử dụng term Customer cho tên class chứ không dung Client.

*Rất nhiều developers gặp lỗi tạo ra các generic terms cho các concepts trong khi đã có những perfectly good terms trong industry/domain.*

1. Các định danh (Identifiers) phải **càng ngắn càng tốt** nhưng không làm lu mờ đi ý nghĩa của nó, nên nằm trong khoảng 20 ký tự trở xuống.
2. **Nên tránh sử dụng các tên gần giống nhau**.  
   Ví dụ: Tên biến persistentObject và persistentObjects không nên sử dụng cùng nhau, cũng như là không nên sử dụng hai tên biến anSqlDatabase và anSQLDatabase
3. **Không nên sử dụng những tên khó hiểu,** kể cả trong trường hợp nó chỉ làm biến đệm hoặc làm biến đếm.  
   *Các tên khó hiểu có thể làm tiêu tốn sức lực của người lập trình khi thời gian tiêu tốn cho việc hiểu xem vai trò của variable/control/method đó hơn là hiểu xem chức năng của nó giải quyết được vấn đề gì.*
4. **Nên tránh viết tắt tên**.  
   Tuy nhiên các nhóm từ đã được quen thuộc, nhiều người biết đến thì nên sử dụng nó. Ví dụ:

computeAverage(); // Không nên: compAvg();  
 generateHTML(); // Không nên: generateHypertextMarkupLanguage();

1. Tên method không nên có bất cứ một **ký tự đặc biệt** nào ngoài dấu gạch dưới.  
   Chỉ sử dụng **dấu gạch dưới** trong trường hợp đặt tên các hằng số

**1.6.2. Class/Interface**

Tuân theo quy ước chung (xem phần 1.6.1), chữ cái đầu viết hoa

* Các Exception classes nên thêm chữ Exception ở cuối, ví dụ: LMSFunctionalException.
* Các Interfaces không có method nào thì nên thêm chữ I ở đầu, ví dụ: IConstants.
* Các Abstract classes nên thêm chữ Abstract ở đầu, ví dụ: abstract class AbstractBean.
* Các Implementation classes nên thêm chữ Impl ở đầu, ví dụ: class CustomerBOImpl implements CustomerBO.

**1.6.3. Biến số** (Variables)

Tên biến được bắt đầu với một ký tự thường và phải tuân theo quy ước về đặt tên ở trên (xem phần 10.1). Ví dụ: custName.

* Các biến List (thuộc kiểu Collection/List) nên thêm vào cuối chữ List, ví dụ: Collection custList.
* Các biến Set (thuộc kiểu Set/HashSet) nên thêm vào cuối chữ Set, ví dụ: Set custSet = new HashSet();
* Các biến Map (thuộc kiểu Map/HashMap/TreeMap) nên thêm vào cuối chữ Map, ví dụ: Map custMap = new TreeMap();
* Các biến Array có thể thêm vào cuối chữ Array, ví dụ: int[] custIDArray;

**1.6.4. Hằng số** (Constants)

Khi đặt tên hằng số phải tuân theo các quy ước sau đây:

* Tuân theo các quy ước đặt tên chung (xem phần 1.6.1).
* Mọi hằng số được tên phải được viếtung hoa tất cả các chữ và giữa các từ được liên kết với nhau bằng dấu gạch dưới.

**1.6.5. Phương thức** (Methods)

Khi đặt tên các method trong class file phải tuân theo các quy ước sau:

* Tuân theo các quy ước đặt tên chung (xem phần 10.1).
* Tên method phải bắt đầu bằng một chữ viết thường.
* Từ đâu tiên của tên method nên sử dụng “động từ”.

Ví dụ:

*/\**

*\* %W% %E% Firstname Lastname*

*\**

*\* Copyright (c) 1993-1996 Sun Microsystems, Inc. All Rights Reserved.*

*\**

*\* This software is the confidential and proprietary information of Sun*

*\* Microsystems, Inc. ("Confidential Information"). You shall not*

*\* disclose such Confidential Information and shall use it only in*

*\* accordance with the terms of the license agreement you entered into*

*\* with Sun.*

*\**

*\* SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF*

*\* THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED*

*\* TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A*

*\* PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR*

*\* ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR*

*\* DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.*

*\*/*

**package** java.blah;

**import** java.blah.blahdy.BlahBlah;

*/\*\**

*\* Class description goes here.*

*\**

*\** ***@author*** *Firstname Lastname*

*\** ***@version*** *1.10 04 Oct 1996*

*\*/*

**public class** Blah **extends** SomeClass {

*/\* A class implementation comment can go here. \*/*

*/\*\**

*\* classVar1 documentation comment*

*\*/*

**public static int** *classVar1*;

*/\*\**

*\* classVar2 documentation comment that happens to be*

*\* more than one line long*

*\*/*

**private static** Object *classVar2*;

*/\*\**

*\* instanceVar1 documentation comment*

*\*/*

**public** Object **instanceVar1**;

*/\*\**

*\* instanceVar2 documentation comment*

*\*/*

**protected int instanceVar2**;

*/\*\**

*\* instanceVar3 documentation comment*

*\*/*

**private** Object[] **instanceVar3**;

*/\*\**

*\* ...method Blah documentation comment...*

*\*/*

**public** Blah() {

*// ...implementation goes here...*

}

*/\*\**

*\* ...method doSomething documentation comment...*

*\*/*

**public void** doSomething() {

*// ...implementation goes here...*

}

*/\*\**

*\* ...method doSomethingElse documentation comment...*

*\**

*\** ***@param someParam*** *description*

*\*/*

**public void** doSomethingElse(Object someParam) {

*// ...implementation goes here...*

}

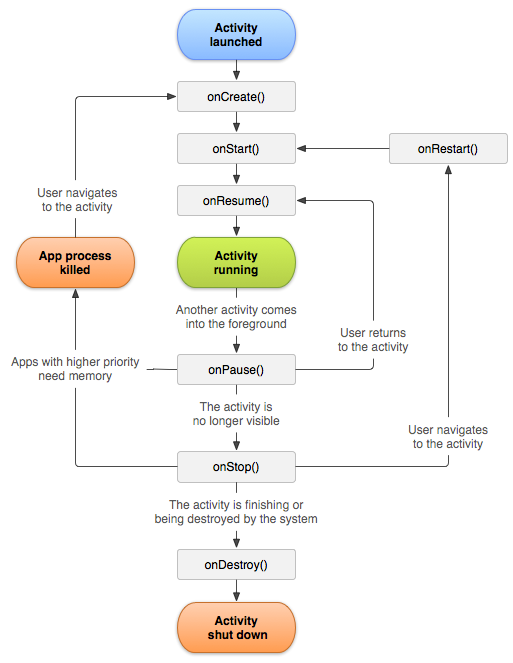
}

**2. Android**

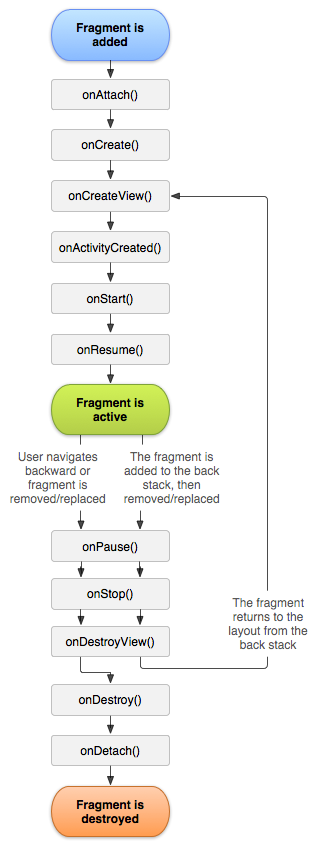
**2.1. Activity** [**[2]**](http://developer.android.com/guide/components/activities.html)

|  |  |
| --- | --- |
| **Callback** | **Description** |
| onCreate() | This is the first callback and called when the activity is first created. |
| onStart() | This callback is called when the activity becomes visible to the user. |
| onResume() | This is called when the user starts interacting with the application. |
| onPause() | The paused activity does not receive user input and cannot execute any code and called when  the current activity is being paused and the previous activity is being resumed. |
| onStop() | This callback is called when the activity is no longer visible. |
| onDestroy() | This callback is called before the activity is destroyed by the system. |
| onRestart() | This callback is called when the activity restarts after stopping it. |

Hình dưới đây mô tả vòng đời của 1 Activity



**2.2. Fragment** [**[3]**](http://developer.android.com/guide/components/fragments.html)



Dưới đây là mô tả về vòng đời fragment

Giai đoạn 1: Khi fragment được tạo, sẽ đi qua các trạng thái sau:

+ onAttach()

+ onCreate()

+ onCreateView()

+ onActivityCreated()

Giai đoạn 2: Khi fragment hiển thị

+ onStart()

+ onResume()

Giai đoạn 3: Khi fragment chạy background:

+ onPaused()

+ onStop()

Giai đoạn 4: Khi fragment bị hủy

+ onPaused()

+ onStop()

+ onDestroyView()

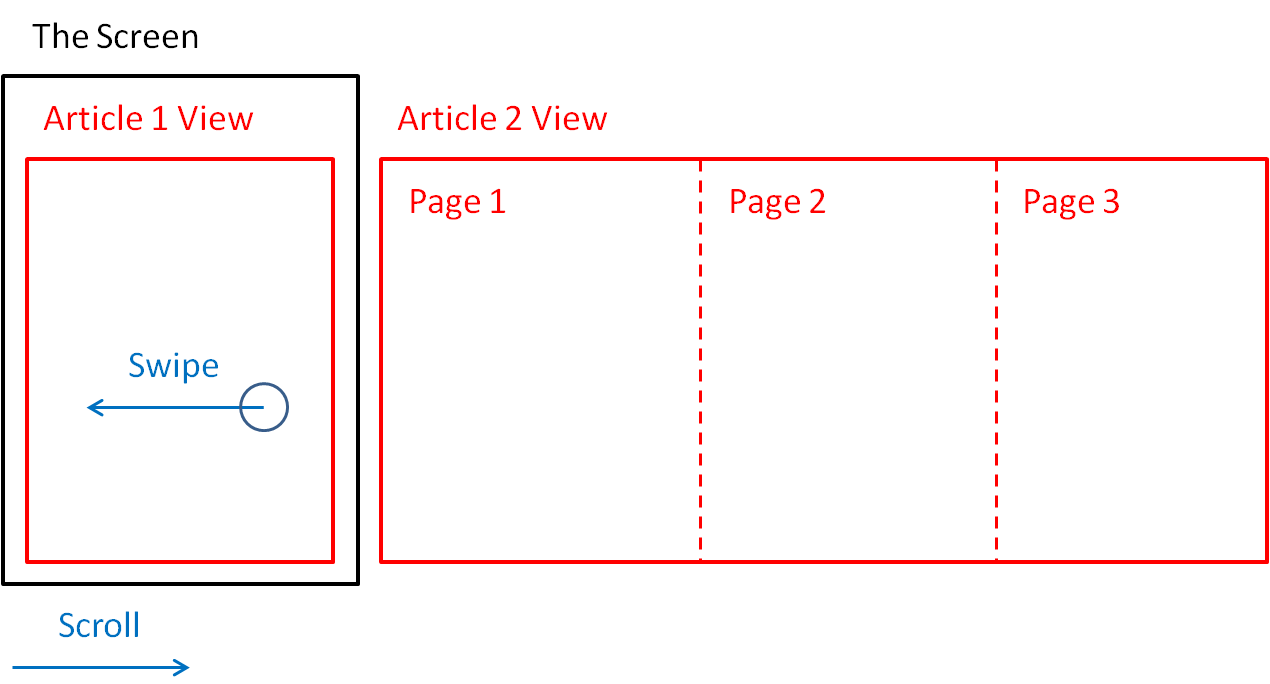
+ onDestroy()

+ onDetach()

\* [ViewPager](http://developer.android.com/reference/android/support/v4/view/ViewPager.html)

Thường được sử dụng như phân thành các trang màn hình. ViewPager thường được kết hợp với

Fragment để tạo dựng.



\* [Task and Back Stack (launch mode)](http://developer.android.com/guide/components/tasks-and-back-stack.html" \l "TaskLaunchModes)

Các loại launch mode:

1. “**standard**” : Đây là chế độ mặc định. Hệ thống tạo ra 1 thể hiện mới của activity trong

task mà từ đó nó đã được start và route intent cho nó. Activity có thể được khởi tạo nhiều lần, mỗi thể hiện có thể thuộc về các task khác nhau và 1 task có nhiều thể hiện.

1. “**singleTop**”: Nếu 1 thể hiện của activity đã tồn tại ở top của Task hiện tại, hệ thống route intent cho thể hiện đó thông qua 1 lời gọi đến phương thức **onNewIntent()** của nó, thay vì tạo ra 1 thể hiện mới của activity này. Activity có thể được khởi tạo nhiều lần, mỗi thể hiện có thể thuộc về các task khác nhau và 1 task có thể có nhiều thể hiện (nhưng chỉ khi activity ở top của back stack không phải là 1 thể hiện đang tồn tại của activity này)

Lưu ý: Khi 1 thể hiện của 1 activity được tạo ra, user có thể nhấn phím BACK để trở lại activity trước đó. Nhưng khi 1 thể hiện của 1 activity xử lý 1 intent mới, user không thể nhấn phím BACK để trở về trạng thái của activity trước khi intent đã đến trong **onNewIntent()**

1. “**singleTask**”: Hệ thống tạo ra 1 task mới và khởi tạo activity tại root của task mới. Tuy nhiên, nếu 1 thể hiện của activity đã tồn tại trong 1 task riêng biệt, hệ thống route intent vào thể hiện đang tồn tại thông qua 1 lời gọi đến **onNewIntent()** của nó, thay vì tạo ra 1 thể hiện mới. Chỉ có 1 thể hiện của activity có thể tồn tại 1 thời điểm.

Lưu ý: Mặc dù activity start trong 1 task mới, phím BACK vẫn trả về cho user 1 activity trước đó.

1. “**singleInstance**”: Cũng giống như “**singleTask**”, ngoại trừ hệ thống không launch bất kì các activity khác vào 1 task đang nắm giữ thể hiện. Activity luôn single và chỉ là member của task của nó; bất kì activity được start ở chế độ này đều mở 1 task rieeng biệt

**2.3. Theme/Style**

Theme/Style là 1 tập các thuộc tính của layout hay window. Các thuộc tính như: chiều cao, cỡ chữ, màu chữ,...

Chi tiết bạn xem ở đây để hiểu rõ hơn <http://developer.android.com/guide/topics/ui/themes.html>

**2.4. UI Layout**

**2.4.1. FrameLayout**[**[4]**](http://developer.android.com/reference/android/widget/FrameLayout.html)

- Khi các view được đưa vào thì sẽ được neo vị trí ở góc trái phía trên màn hình. FrameLayout không cho phép thay đổi vị trí của các view.

- Khi đưa view vào sau sẽ đè hay che 1 phần view trước đó

**2.4.2. LinearLayout** [[5]](http://developer.android.com/guide/topics/ui/layout/linear.html)

- Layout này cho phép sắp xếp các view theo chiều ngang hay dọc

- Có thể chia height/width theo tỉ lệ cho các view con của LinearLayout

**2.4.3. RelativeLayout** [**[6]**](http://developer.android.com/guide/topics/ui/layout/relative.html)

Cho phép sắp xếp các view theo 1 vị trí tương đối so với các view khác. RelativeLayout dựa vào id của các view để sắp xếp.

Lưu ý nhỏ khi muốn tối ưu giao diện: Về tốc độ render của RelativeLayout luôn nhanh hơn LinearLayout.

Chi tiết bạn đọc trong: <http://developer.android.com/training/improving-layouts/optimizing-layout.html>

**2.5. UI Controls**

**2.5.1. ListView** [**[7]**](http://developer.android.com/guide/topics/ui/layout/listview.html)

Điều đầu tiên bạn nên dùng RecyclerView (2.5.3) thay thế cho Listview. Lý do tôi sẽ ghi trong mục 2.5.3. RecyclerView

Trong ứng dụng cần lưu trữ và hiển thị danh sách thông tin thì chúng ta có thể dùng ListView



List items sẽ được tự động cho vào danh sách của ListView thông qua Adapter

\* ViewHolder:

[http://developer.android.com/training/improving-layouts/smooth-scrolling.html#ViewHolder](http://developer.android.com/training/improving-layouts/smooth-scrolling.html" \l "ViewHolder)

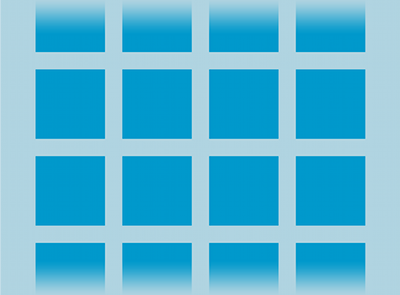
Mục đích để tối ưu giao diện. Tránh việc findById() quá nhiều lần.

**2.5.2. GridView** [**[8]**](http://developer.android.com/guide/topics/ui/layout/gridview.html)

Điều đầu tiên bạn nên dùng RecyclerView (2.5.3) thay thế cho GridView. Lý do tôi sẽ ghi trong mục 2.5.3. RecyclerView

Nếu bạn muốn hiển thị list item thành dạng lưới đều nhau (Grid) bạn có thể dùng GridView

Về cơ bản cách sử dụng GridView khá giống với ListView.



**2.5.3. RecyclerView**[[](https://developer.android.com/training/material/lists-cards.html" \l "RecyclerView)9]

Sẽ có bạn hỏi là có cách nào để gộp được chức năng của ListView và GridView không ?

Tôi xin trả lời là RecyclerView hoàn toàn làm được.

RecyclerView được hỗ trợ trong **android.support.v7**. RecyclerView là 1 phiên bản tiên tiến và linh hoạt hơn ListView

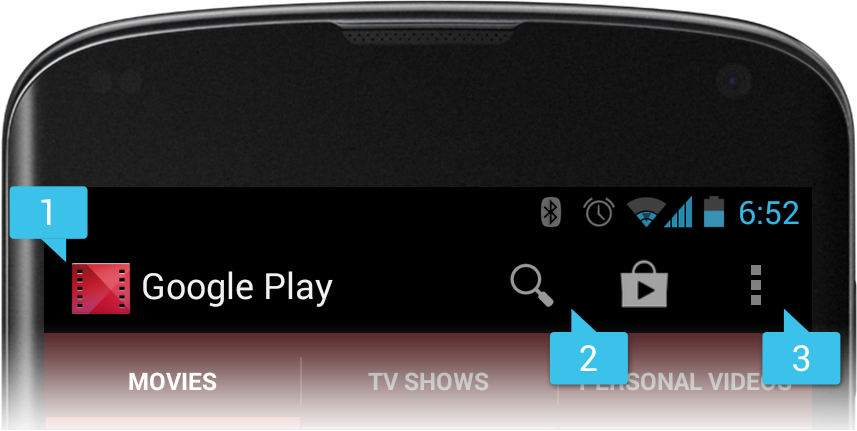
RecyclerView cung cấp 3 dạng hiển thị:

+ LinearLayoutManager : hiển thị các items dạng danh sách dọc/ngang (vertical/horizontal list)

+ GridLayoutManager : hiển thị dưới dạng lưới

+ StaggeredGridLayoutManager: hiển thị dạng lưới so le

**2.5.4. Toolbar** [**[10]**](https://developer.android.com/reference/android/widget/Toolbar.html)

[1] app icon, [2] two action items, [3] action overflow

Nếu trước đây bạn đã sử dụng ActionBar [**[11]**](http://developer.android.com/guide/topics/ui/actionbar.html) thì giờ bạn nên chuyển sang dùng ToolBar. Sau 1 thời gian dùng ToolBar bạn sẽ thấy được nhiều thứ ToolBar làm được mà ActionBar không làm được.

Ưu điểm:

+ Dễ dàng điều hướng (navigation button) app

+ Dễ dàng customs layout.

+ Có thể đặt bất kì đâu trong layout (đối với actionBar thì chỉ có thể đặt ở phía trên cùng)

**2.6. Service** [[12]](http://developer.android.com/guide/components/services.html)

Service là thành phần ứng dụng chạy ngầm và không được cung cấp giao diện người dùng.

Các method sau nên override:

* **onStartCommand()**: method này được gọi khi các thành phần khác (như activity) yêu cầu chạy service bằng cách gọi ***startService()***. Chú ý phải gọi ***stopSeft()*** hay ***stopService()*** khi không cần chạy Service nữa
* **onBind()** : method này được gọi khi Component khác muốn kiên kết với Service
* onCreate(): method này được gọi khi lần đầu tiên được chạy. Nếu service đang chạy thì method này sẽ không được gọi.
* **OnDestroy()**: method này sẽ được gọi khi Service không còn được sử dụng nữa. Lúc này bạn cần phải hủy mọi Resouces như: Thread, Listeners, Receivers, …

Cũng như Activity, để service có thể chạy được thì bạn cần khai báo service trong Manifest

<manifest ... >

...

<application ... >

<service android:name=".ExampleService" />

...

</application>

</manifest>

**Chú ý:**

+ Các hệ thống Android sẽ buộc dừng Service khi còn rất ít bộ nhớ và nó sẽ tập trung resource cho activity nào đang được user sử dụng. Nếu service và activity đang được User sử dụng thì ít khả năng bị kill Service và nếu service được khai báo chạy **[Fore](http://developer.android.com/guide/components/services.html" \l "Foreground)ground** thì service này sẽ không bao giờ bị hủy.

+ Trường hợp còn lại, nếu servive được chạy và chạy lâu thì System sẽ hạ thấp vị trí trong danh sách Background Task theo thời gian và khả năng cao Service sẽ bị hủy → **Bạn cần thiết kế code cho việc System hủy và khởi động lại Service của bạn.**

Có 2 class có thể kế thừa để khởi tạo Service:

+ **Service:** là class cơ bản cho Service. Service dùng Thread chính của ứng dụng nên có thể làm chậm ứng dụng

+ **IntentService**: đây là lựa chọn tốt nếu bạn không cần Service xử lý nhiều yêu cầu cùng 1 lúc

\* **IntentService** [[](http://developer.android.com/guide/components/services.html" \l "ExtendingIntentService)13]

+ Tạo 1 hàng đợi để gửi 1 intent vào 1 thời điểm cho ***onHandleIntent()***. Vậy nên không cần lo về xử lý đa luồng

+ Ngừng chạy Service sau khi tất cả request đã được xử lý. Vậy nên không cần phải gọi ***stopSeft()***

***+*** Cung cấp mặc định ***onBind()*** trả về null

+ Cung cấp mặc định ***onStartCommand*()** để gửi các intent vào hàng đợi công việc và sau đó thực hiện ***onHandleIntent*()**

**2.7. Database** [[14]](http://developer.android.com/training/basics/data-storage/databases.html)

SQLite là hệ thống cơ sở dữ liệu quan hệ nhỏ gọn, hoàn chỉnh, có thể cài đặt bên trong các trình ứng dụng khác.

Ưu điểm:

- Tin cậy

- Không cần cài đặt cấu hình

- Kích thước nhỏ gọn với cấu hình đầy đủ không đầy 300KB

- Không cần phần mềm phụ trợ

**2.8. Thread**

**2.8.1. AsyncTask** [[15]](http://developer.android.com/reference/android/os/AsyncTask.html)

***AsyncTask*** cho phép sử dụng hợp lý và dễ dàng cho UI Thread. **AsyncTask** cho phép thực hiện các hành động ở background và đưa kết quả lên UI Thread mà không cần thao tác với **Thread** và/hoặc **Handler.**

AsyncTask's generic types: (xin phép được giữ nguyên cụm từ này. Để hiểu rõ hơn về **Generic** xin hãy ấn vào [đây](http://www.tutorialspoint.com/java/java_generics.htm))

**AsyncTask<Params,Progress,Result>**

1. Params: là kiểu dữ liệu để truyền vào AsyncTask. Cụ thể là cho method **doInBackground(Params...)**
2. Progress: là kiểu dữ liệu để update giá trị cho method **onProgressUpdate(Progress...)**
3. Result: là kiểu dữ liệu trả về khi **doInBackground()** kết thúc và sẽ được truyền vào param của method **onPostExecute(Result)**.

**4 bước** cơ bản của AsyncTask:

1. ***onPreExecute()***: được gọi trước khi Task được thực hiện
2. ***doInBackground(Params…)***: các task chính hay công việc xử lý ngầm và được gọi sau **onPostExecute()** (lưu ý: không dùng UI Control ở đây vì sẽ gây lỗi. Nếu không tin thì bạn có thể thử ^\_^)
3. ***onProgressUpdate(Progress…)***: ở đây sẽ update giá trị cho UI ví dụ như update giá trị phần trăm. Method này được gọi sau khi gọi **publishProgress(Progress...)**
4. ***onPostExecute(Result)***: được gọi sau khi các xử lý trong **doInBackground()** kết thúc.

**2.8.2. Handler** [[16]](http://developer.android.com/reference/android/os/Handler.html)

Tôi thì thường sử dụng Handler để chạy 1 Task sau 1 khoảng thời gian định trước

**2.8.3. Thread** [[17]](http://developer.android.com/reference/java/lang/Thread.html)

**Tài liệu tham khảo**

[1] Java Code Conventions : <http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

[2] Activity: <http://developer.android.com/guide/components/activities.html>

[3] Fragment: <http://developer.android.com/guide/components/fragments.html>

[4] FrameLayout: <http://developer.android.com/reference/android/widget/FrameLayout.html>

[5] LinearLayout: <http://developer.android.com/guide/topics/ui/layout/linear.html>

[6] RelativeLayout: <http://developer.android.com/guide/topics/ui/layout/relative.html>

[7] ListView: <http://developer.android.com/guide/topics/ui/layout/listview.html>

[8] GridView: <http://developer.android.com/guide/topics/ui/layout/gridview.html>

[9] RecyclerView: [https://developer.android.com/training/material/lists-cards.html#RecyclerView](https://developer.android.com/training/material/lists-cards.html" \l "RecyclerView)

[10] Toolbar: <https://developer.android.com/reference/android/widget/Toolbar.html>

[11] ActionBar: <http://developer.android.com/guide/topics/ui/actionbar.html>

[12] Service: <http://developer.android.com/guide/components/services.html>

[13] IntentService: [http://developer.android.com/guide/components/services.html#ExtendingIntentService](http://developer.android.com/guide/components/services.html" \l "ExtendingIntentService)

[14] Database: <http://developer.android.com/training/basics/data-storage/databases.html>

[15] AsyncTask: <http://developer.android.com/reference/android/os/AsyncTask.html>

[16] Handler: <http://developer.android.com/reference/android/os/Handler.html>

[17] Thread: <http://developer.android.com/reference/java/lang/Thread.html>