**1.Activity**

Một Activity được hiểu như là một màn hình thể hiện những gì đang diễn ra đang thao tác với giao diện người dùng (user interface).

**1.1. Avtivity Lifecycle**



- onCreate() được gọi lần đầu tiên khi activity bắt đầu tạo

- onStart() được gọi khi activity đắt đầu thực hiện cho người dùng thấy

- onResum() khi activity bắt đầu có sự tương tác với người dùng

- onPause() được gọi khi activity tạm dừng và trước đó là đã được resumed

- onStop() được gọi khi activity không còn hiện cho người dùng, tức là việc tương tác của người dùng với chương trình sẽ không còn (chuyển activity khác, nhấn phím home)

- onDestroy() khi activity bị hủy

- onRestart() khi activity bị hủy và được gọi lại restartung again

**1.2. Styles and Themes**

***Styles***

Android Style làm việc tương tự như CSS. Một Style xác định định dạng một user interface, nó có thể được áp dụng cho một individual View(từ bên trong một file Layout) hoặc một Activity, hoặc úng dụng (từ bên trong file Manifest).

Định dạng Styles: Một Style được định nghĩa trong file styles.xml (res/values/styles.xml).

*<?xml version="1.0" encoding="utf-8"?>*

*<resources>*

*<style name="CustomFontStyle">*

*<item name="android:layout\_width">fill\_parent</item>*

*<item name="android:layout\_height">wrap\_content</item>*

*<item name="android:capitalize">characters</item>*

*<item name="android:typeface">monospace</item>*

*<item name="android:textSize">12pt</item>*

*<item name="android:textColor">#00FF00</item>/>*

*</style>*

*</resources>*

hoặc ta có thể kế thừa từ Styles khác

*<resources>*

*...*

*<style name="MyCustomTheme" parent="android:style/Theme">*

*<item name="android:textColorPrimary">#ffff0000</item>*

*</style>*

*...*

*</resources>*

***Themes***

Theme là style nhưng được áp dụng cho toàn bộ một avtivity hay cho cả ứng dụng.

*<application android:theme="@style/CustomFontStyle">*

*<activity android:theme="@style/CustomFontStyle">*

**2. Fragment**

Fragment là một thành phần giao diện người dùng hoặc một hành vi của ứng dụng. Fragment có thể được cài đặt trong Activity, nó có thể cho phép thiết kế activity với nhiều module. Có thể nói fragment là một loại sub-Activity.

- Fragment cũng có layout riêng, có các hành vi và vòng đời riêng

- Chúng ta có thể thêm hoặc xóa Fragment trong một Activity trong khi Activity này đang chạy

- Có thể kết hợp nhiều Fragment trong một Activity để xây dựng giao diện người dùng đa khung

- Một Fragment có thể được sử dụng trong nhiều Activities

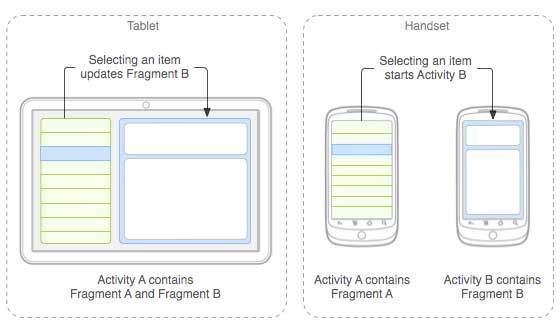
- Vòng đời của Fragment có quan hệ chặt chẽ với vòng đời của Activity đang dùng nói, điều này có nghĩa là khi Activity bị tạm dừng thì các Fragment sẽ dừng lại

- Fragment có thể thực hiện một hành vi mà không có trong thành phần user interface

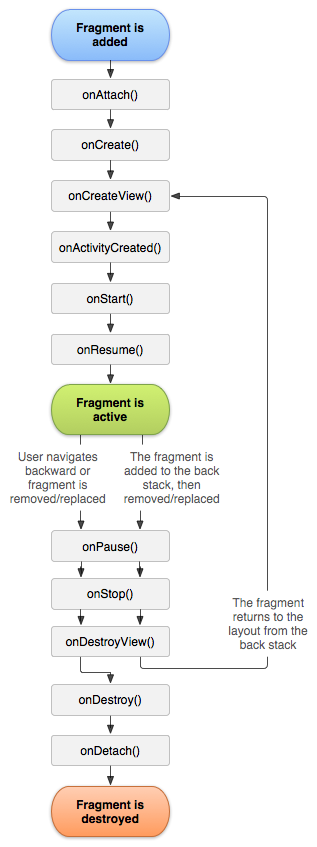
- Fragment được thêm vào từ API11 trở lên

- Có thể tạo các Fragment bằng cách kế thừa lớp Fragment và Fragment được thêm vào layout bời thẻ <fragment>

Cùng một thời điểm chúng ta chỉ có thể hiển thị một Activity duy nhất trên màn hình, vì vậy ta không thể chia màn hình thiết bị ra thành nhiều phần và kiểm soát các thành phần khác nhau này một cách riêng biệt. Nhưng với Fragment thì màn hình được linh hoạt hơn, xóa bỏ việc chỉ có duy nhất một Activity duy nhất, nhưng Activity này có thể chứa nhiều Fragment với layout, event, và lifecycle riêng.



**2.1. Life Cycle**

Fragment có vòng đời rất giống với vòng đời của một Activity.

Giai đoạn 1: Fragment được khởi tạo

1. onAttach()

2. onCreate()

3.onCreateView()

4.onActivityCreated()

Giai đoạn 2: Fragment được hiển thị

1. onStart()

2. onResum()

Giai đoạn 3: Fragment chạy ẩn (background mode)

1. onPause()

2. onStop()

Giai đoạn 4: Fragment bị destroy

1. onPause()

2. onStop()

3. onDestroyView()

4. onDestroy()

5. onDectach()

**2.2. Methods**

- onCreate(): hệ thống gọi phương thức này khi tạo Fragment. Bạn nên khởi tạo các thành phần thiết yếu của Fragment mà bạn muốn giữ lại khi Fragment được tạm dừng hoặc sau đó được tiếp tục.

- onCreateView(): hệ thống gọi phương thức này khi Fragment vẽ giao diện của chính nó lần đầu tiên. Để vẽ giao diện cho Fragment bạn cần phải trả về một thành phần View từ phương thức này. Trả về null nếu Fragment không cung cấp giao diện.

- onPause(): hệ thống gọi phương thức này như là để đánh dấu lần đầu người dùng rời Fragment.

**3. Services**

Service là một trong bốn compoment cơ bản của android (services, antivities, content providers, broadcast receivers).

Service chạy ẩn ở dưới để thực hiện các thao tác mà không cần tương tác người dùng.

Một service có 2 loại:

- Started: một Service được bắt đầu giống như compoment khác như activities, bắt đầu service bằng cashc gọi phương thức startService(). Thông thường, một “start service” thực hiện một hành động đơn lẻ và không trả về kết quả cho đối tượng.

- Bound: một service được ràng buộc khi các component ràng buộc thông qua phương thức bindService(). Service ràng buộc thường là kiểu giao diện client-server, nó cho phép các component tương tác với service, gửi yêu cầu, nhận kết quả trả về. Một service ràng buộc có thể chạy với nhiều component ràng buộc đến nó khi chạy lần đầu tiên, khi tất cả các component của chúng không còn ràng buộc nữa thì service sẽ bị hệ thống hủy.



Để tạo một service bạn cần tạo một class kế thừa từ class Service hoặc một subclass. Một số phương thức quan trọng:

- onStartCommand(): hệ thống sẽ gọi hàm này khi một component khác hoặc một activity yêu cầu bắt đầu service bằng gọi phương thức startService(). Nếu bạn chỉ muốn sử dụng service kiểu ràng buộc thì không cần phải cài đặt phương thức này.

- onBind(): hệ thống sẽ gọi hàm này khi một component muốn ràng buộc mới service bằng cách gọi phương thức bindService(). Khi cài đặt hàm này bạn phải cung cấp giao tiếp giữa client và service bằng cách trả về iBinder. Bạn luôn luôn phải cài đặt hàm này, nếu không muốn ràng buộc thì có thể trả về null.

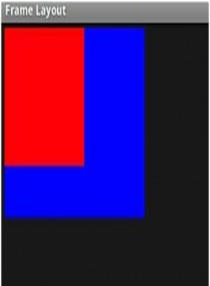
- onDestroy(): hệ thống gọi hàm này khi service không được sử dụng nữa và cần phải gọi phương thức này để giải phóng tài nguyên cần thiết liên quan đến service.

**4. UI Layout**

**4.1. FrameLayout**

- là loại layout cơ bản nhất, đặc điểm của nó là khi gắn các contral lên giao diện thì các control này sẽ luôn được “neo” ở góc trái trên màn hình, nó không cho phép chúng ta thay đổi vị trí của các control theo một location nào đó.

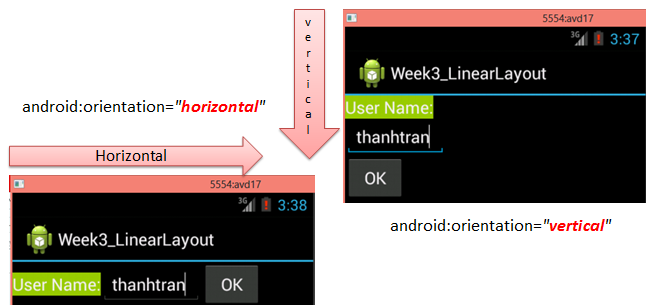
- các control đưa vào sau đó sẽ đè lên trên và che khuất control trước đó (trừ khi ta thiết lập transparent cho control sau)



**4.2. LinearLayout**

- Layout này cho phép sắp xếp các control theo 2 hướng trên giao diện: Hướng từ trái qua phải và hướng từ trên xuống dưới.

- có thể dùng margin, gravity, weight,... để hỗ trợ cho việc thiết kế.

****

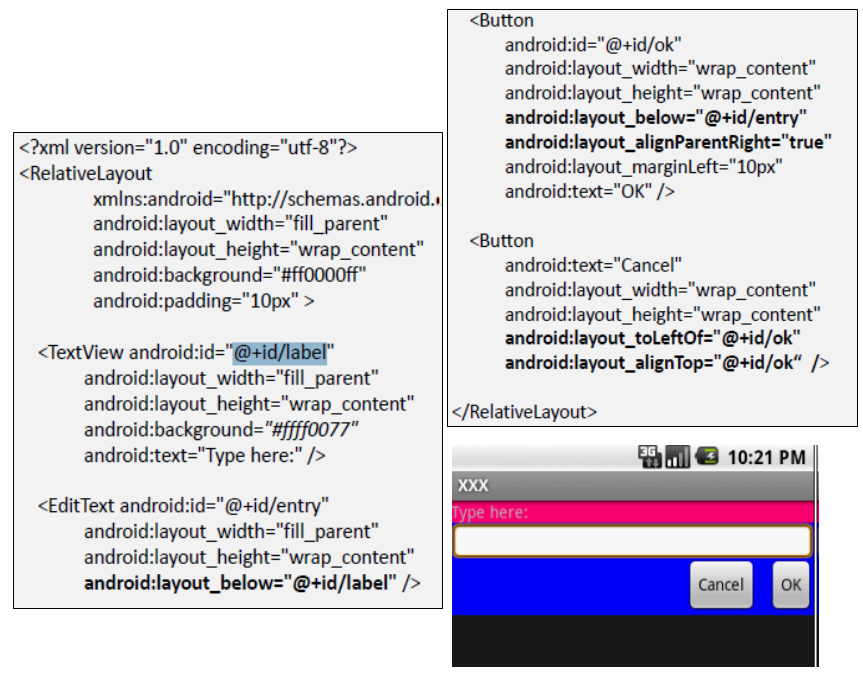
**4.3. TableLayout**

- là layout cho phép sắp xếp các control theo dạng lưới (hàng – cột).

- TableLayout sẽ xem dòng nào có số lượng control nhiều nhất để xác định rằng nó có bao nhiêu cột (lấy dòng có số lượng control nhiều nhất làm số cột chuẩn).

**4.4. RelativeLayout**

- RelativeLayout cho phép sắp xếp các control theo vị trí tương đối giữa các control khác trên giao diện (kể cả control chứa nó). Thường nó dựa vào Id của các control khác để sắp xếp theo vị trí tương đối. Do đó kho làm RelativeLayout phải chú ý là đặt Id control cho chuẩn xác, nếu sau khi Layout xong mà thay đổi lại Id của các control thì giao diện sẽ bị xáo trộn (do đó nếu đổi Id thì phải đổi luôn cá tham chiếu khác sao cho khớp với Id mới đổi).



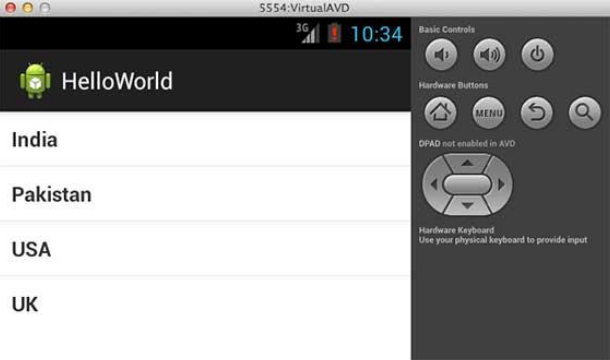
**4.5. AbsoluteLayout**

- là layout cho phép thiết lập các control giao diện theo vị trí tùy thích.

- sử dụng layout\_x, layout\_y đẻ xác định vị trí của control.

**4.6. ListView**

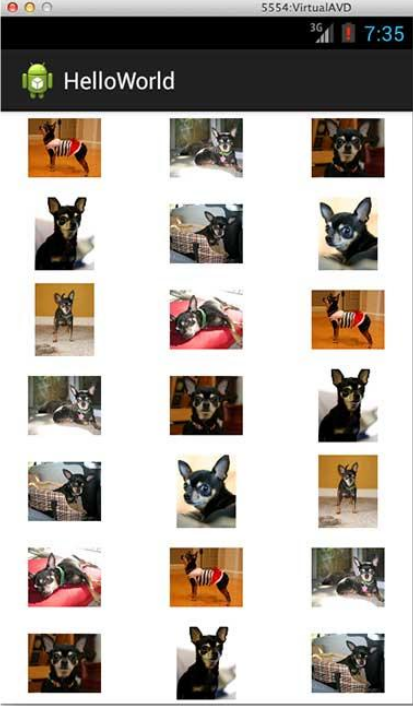
- các items được nhóm và hiển thị trong một danh sách cuộn theo chiều dọc. Các item được chèn tự động vào danh sách bằng cách sử dụng Adapter để đẩy content từ một mảng hay từ database.



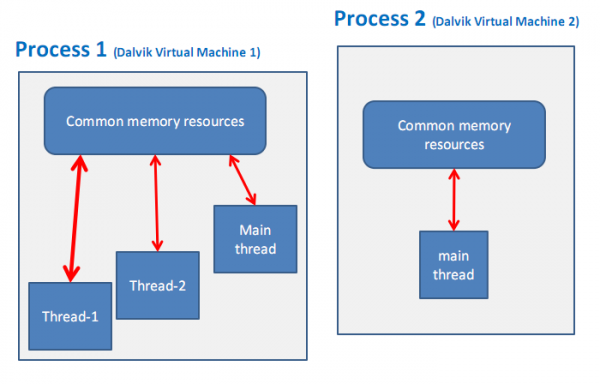
**4.7. GridView**

- các items được thể hiện trong khung lưới 2 chiều theo hàng và cột, lưới các item không nhất thiết phải xác định trước, nhưng các items sẽ được tự động chèn vào layout sử dụng một ListAdapter.

- ListView và GridView điều là các subclass của AdapterView.



**5. Multi-thread**



Trong ứng dụng android, ta có thể khởi chạy đa luồng bất cứ lúc nào mà chúng ta muốn (sau một sự kiện hoặc do một sự kiện nào đó tự kích hoạt,... ), thường thì sẽ có một luồng chính và nhiều luồn phụ chạy song song đồng thời và lúc nào đó các luồng phụ sẽ can thiệt vào luồng chính thông qua các biến toàn cục.

Về cơ bản có 2 cách chính để thực thi luồng (Thread) trong một đoạn code của ứng dụng:

- Tạo một class extends lớp Thread và override hàm run().

- Khởi tạo một đối tượng mới là thể hiện của luồng thông qua đối tượng Runnable.

Cả 2 cách này đều phải gọi phương thức start() để bắt đầu thực thi luồng mới tạo.

Một ứng dụng có thể có một hoạt động tốn thời gian, tuy nhiên ta muốn UI vẫn đáp ứng tốt đối với các tương tác của người dùng. Android cung cấp 2 cách để xử lý tình huống này:

- Thực hiện thao tác đó trong một service ở background và dùng notification để thông báo cho người dùng bước tiếp theo.

- Thực hiện tao tác đó trong một background thread. Các thread của Android tương tác với nhau bằng cách sử dụng các đối tượng Handler và post các đối tượng Runnable tới view chính.

Handler Class:

- Khi một tiến trình được tao cho một ứng dụng, main thread của nó được dành riêng để chạy một message queue, queue này quản lý các đối tượng bậc cao của ứng dụng (activity, intent recevier, … ) và các của sổ mà chúng tạo ra.

- Ta có thể tạo các thread phụ, chúng tương tác với thread chính của ứng dụng qua một Handler.

- Khi ta tạo một Handler mới, nó được gắn với message của thread tạo ra nó - từ đó trở đi, nó sẽ gửi các message và các runnable tới message queue đó và tực thi chúng khi chúng ra khỏi message queue.

AsyncTask Class: cho phép người sử dụng UI thread một cách dễ dàng và đúng cách. AsyncTask cho phép thực hiện các hoạt động background và gửi kết quả cho UI thread mà không phải thao tác với thread hoặc handler. Một tác vụ không đồng bộ là một nhiệm vụ tính toán chạy tại một background thread và kết quả được gửi cho UI thread.

**6. Database**

Lưu trữ dữ liệu là tính năng quan trọng đối với ứng dụng, giúp cho người dùng có thể dùng lại được những dữ liệu trước đó mà không caanfnhapaj lại. Trong Android có 3 cách để lưu lại dữ liệu:

- Cơ chế “cấu hình chia sẻ” (shared preferences) được dùng để lưu những dữ liệu nhỏ dưới dạng key-value.

- Lưu dữ liệu cố định vào tệp tin trong bộ nhớ trong hoặc bộ nhớ ngoài của điện thoại.

- Lưu dữ liệu sử dụng cơ sở dữ liệu quan hệ cục bộ SQLite

**6.1. Lưu trữ dữ liệu cố định với shared preferences**

Android cung cấp sẵn một cơ chế đơn giản giúp chúng ta lưu trữ nhanh các dữ liệu ngắn như cấu hình ứng ứng dụng, tên đăng nhập, email, … và lấy lại dữ liệu đã ghi này trong các lần chạy ứng dụng tiếp theo. Để thực hiện việc này ta sử dụng một loại Activity đặc biệt là Preferences được cung cấp sẵn.

Để sử dụng đc PreferencesActivity, trước hết các thông tin cần lưu trữ phải được mô tả trong một file xml, sau đó tạo 1 Activity kế thừa từ PreferenceActivity:

*<?xml version="1.0" encoding="utf-8"?>*

*<PreferenceScreen*

*xmlns:android="http://schemas.android.com/apk/res/android">*

*<PreferenceCategory android:title="Category 1">*

*<CheckBoxPreference*

*android:title="Checkbox"*

*android:defaultValue="false"*

*android:summary="True or False"*

*android:key="checkboxPref" />*

*</PreferenceCategory>*

*<PreferenceCategory android:title="Category 2">*

*...*

*...*

*<PreferenceScreen*

*...*

*</PreferenceScreen>*

*</PreferenceCategory>*

*</PreferenceScreen>*

----

*public class AppPreferenceActivity extends PreferenceActivity {*

*@Override*

*public void onCreate(Bundle savedInstanceState) {*

*super.onCreate(savedInstanceState);*

*PreferenceManager prefMgr = getPreferenceManager();*

*prefMgr.setSharedPreferencesName("appPreferences");*

*//---load the preferences from an XML file---*

*addPreferencesFromResource(R.xml.myapppreferences);*

*}*

*}*

Việc lưu trữ dữ liệu là trong suốt với người dùng. Trên thực tế, các thông tin được lưu trong 1 file xml nằm trên bộ nhớ trong, trong vùng nhớ chỉ có thể truy cập được bởi chính ứng dụng tạo ra nó.

**6.2. Lưu trữ dữ liệu bằng file trên bộ nhớ trong và bộ nhớ ngoài**

Trong trường hợp cần lưu trữ dữ liệu phức tạp hơn, ta có thể sử dụng các lớp nhập xuất file có sẵn của gói java.io

**6.3. Cơ sở dữ liệu SQLite**

Đối với loại dữ liệu quan hệ thì việc sử dụng cơ sở dữ liệu quan hệ sẽ thuận tiện hơn rất nhiều. Ví dụ, ta cần lưu trữ kết quả kiểm tra của các sinh viên trong trường học. Dùng cơ sở dữ liệu sẽ cho phép chúng ta truy vấn kết quả của tập sinh viên nhất định theo các tiêu chí khác nhau. Việc thêm, bớt, thay đổi thông tin thông qua các câu truy vấn SQL cũng dễ dàng hơn nhiều so với việc thao tác trên file. Android sử dụng hệ cơ sở dữ liệu SQLite. CSDL do một ứng dụng tạo ra sẽ chỉ được truy xuất bởi ứng dụng đó, và file CSDL sẽ nằm trong bộ nhớ trong dành riêng cho ứng dụng.

Một thói quen tốt thường được các lập trình viên kinh nghiệm sử dụng là tập trung tất cả mã lệnh truy cập đến CSDL vào một lớp riêng để thao tác trên CSDL trở nên trong suốt với môi trường ngoài. Chúng ta sẽ tạo trước một lớp như vậy, gọi là DBAdapter.

Trước tiên ta khai báo các hằng số như: tên CSDL, tên bảng, tên các trường để dễ dàng truy xuất và thay đổi trong quá trình phát triển. Ngoài ra, ta cũng khai báo phiên bản (do ta tự đánh số) của CSDL trong ứng dụng và viết sẵn câu truy vấn dùng để tạo CSDL:

*static final String KEY\_ROWID = "\_id";*

*static final String KEY\_NAME = "name";*

*static final String KEY\_EMAIL = "email";*

*static final String TAG = "DBAdapter";*

*static final String DATABASE\_NAME = "MyDB";*

*static final String DATABASE\_TABLE = "contacts";*

*static final int DATABASE\_VERSION = 2;*

*static final String DATABASE\_CREATE =*

*"create table contacts (\_id integer primary key autoincrement, "*

*+ "name text not null, email text not null);";*

Ngoài ra ta cũng viết thêm các hàm, phương thức để thực hiện các truy vấn CSDL như mở CSDL, tạo mới bảng ghi, cập nhật bảng ghi, lấy tất cả bảng ghi, lấy bảng ghi theo id, ...

**Java Convention**

**1. Tổ chức File**

Một file bao gồm nhiều phần, các phần này nên được tách biệt nhau bởi các dòng trống và các chú thích không bắt buộc để xác định mỗi phần.

File có độ dài hơn 2000 dòng hoặc có nhiều hơn 50 phương thức sẽ rất nặng nề và nên tránh.

***File mã nguồn Java***

Mỗi file mã nguồn Java chứa một public class đơn hoặc một interface. Nếu các private class và interface được kết hợp với một public class, bạn có thể đặt chúng trong cùng file mã nguồn dưới dạng một public class. Public class nên là class hoặc interface đầu tiên trong file.

File mã nguồn Java có thứ tự như sau :

* Beginning comments
* Package and Import statements
* Class and interface declarations

***Beginning Comments***

Tất cả các file mã nguồn của một dự án nên có chung định dạng của chú thích ban đầu, chứa các thông tin như tên class, thông tin phiên bản, thời gian, ghi chú về copyright, các log về sự thay đổi mã nguồn …

*/\**

*\* Classname*

*\**

*\* Version info*

*\**

*\* Copyright notice*

*\*/*

***Package and Import Statements***

Câu lệnh pakage là câu lệnh đầu tiên mà không phải là chú thích của hầu hết các file mã nguồn Java.

*package java.awt;*

*import java.awt.peer.CanvasPeer;*

***Class and Interface Declarations***

|  |  |  |
| --- | --- | --- |
| No | Phần khai báo của Class/Interface | Ghi chú |
| 1 | Chú thích tài liệu Class/interface  (/\*\*...\*/) |  |
| 2 | Câu lệnh class or interface |  |
| 3 | 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. |
| 4 | 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. |
| 5 | 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. |
| 6 | 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. |
| 7 | Constructors |  |
| 8 | 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. |
| 9 | Inner classes/interfaces |  |

**2. Indentation and Braces**

## ***Tab và Indent***

Một đơn vị của indent được tính bằng 4 kí tự trống.

Không nên sử dụng kí tự tab vì các trình biên soạn khác nhau thì kí tự này lại khác nhau.

Indent tiếp theo sẽ là 8 kí tự trống ( bằng 2 mức indent thường).

## ***Dấu ngoặc nhọn***

Dấu ngoặc nhọn mở “{“ của khai báo class/ method và các khối lệnh khác nên đặt tại **cuối của dòng lệnh đầu tiên trong khối lệnh đó**.

## ***Độ dài dòng lệnh***

Tránh viết dòng lệnh dài hơn 80 hoặc 120 kí tự vì một số tool sẽ không xử lý tốt được những dòng lệnh này.

**Ghi chú:** Các đoạn code trong các tài liệu nên có độ dài ngắn hơn bình thương, nhỏ hơn 70 kí tự.

## ***Ngắt dòng***

Khi một biểu thức dài hơn một dòng đơn, cần ngắt chúng theo nguyên tắc sau :

* Ngắt sau dấu phẩy
* Ngắt sau toán tử logic
* Ngắt trước toán tử
* Những quy ước ngắt có độ ưu tiên cao hơn sẽ được ưu tiên trước.
* Căn dòng mới được ngắt cùng cấp với dòng trước nó.
* Nếu quy tắc trên dẫn đến sự nhầm lẫn code hoặc code vượt quá lề phải thì sử dụng indent với 8 kí tự trống để thay thế

**3. Comments**

***Block Comments***

*/\**

*\* Here is a block comment with some very special*

*\* formatting that I want indent(1) to ignore.*

*\**

*\* one*

*\* two*

*\* three*

*\*/*

***Single-Line Comments***

*if (condition) {*

*/\* Handle the condition. \*/*

*...*

*}*

***Trailing Comments***

*if (a == 2) {*

*return TRUE; /\* special case \*/*

*} else {*

*return isprime(a); /\* works only for odd a \*/*

*}*

***End-Of-Line Comments***

*if (foo > 1) {*

*// Do a double-flip.*

*...*

*}*

*else*

*return false; // Explain why here.*

*//if (bar > 1) {*

*//*

*// // Do a triple-flip.*

*// ...*

*//}*

*//else*

*// return false;*

**4. Naming Conventions**

## ***General Rules***

Phần này sẽ đưa ra một số quy ước chung về đặt tên Source File / variable / control / method.

1. Người lập trình khi đặt tên phải **có ý nghĩa**, và phải nó phải chỉ ra được mục đích của file/variable/control/method.
2. 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

***Class/Interface***

Tên của mỗi class/interface phải được bắt đầu bằng một chữ hoa và phải tuân theo quy ước chung của việc đặt tên , ví dụ: class CustomerBean.

* Các Exception classes nên thêm chữ Exceptin 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.

***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;

Về việc sử dụng tên ID hay Id thì tùy thuộc vào từng ứng dụng.

***Constants***

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

* Tuân theo các quy ước đặt tên chung (xem phần 10.1).
* Mọi hằng số được tên phải được viết hoa tất cả các chữ và giữa các từ được liên kết với nhau bằng dấu gạch dưới.
* Mọi hằng số phải được khai báo static final.

***Methods***

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

* Tuân theo các quy ước đặt tên chung.
* Tên method phải bắt đầu bằng một chữ viết thường.
* Từ đâu tiên của tên method nên sử dụng “động từ”. Ví dụ đây là một số động từ thường dùng:

getCustomerID, setCustomerID

isActive, hasAddresses

findCustomers, searchCustomers

computeSalary, calculateSalary

initializeParameters, initParameters

addCustomer, removeCustomer