1. Thread và Multithreading
2. Thread

Thread là đơn vị nhỏ nhất của tiến trình được định thời bởi hệ điều hành và được bao hàm trong các tiến trình thực thi của máy tính. Mỗi một thread có một callstack cho các phương thức, đối số và biến cục bộ của thread đó.

Android chạy trên một hệ điều hành nhân Linux. Các ứng dụng của Android được viết bằng ngôn ngữ Java, và chúng chạy trong một máy ảo, đó không phải máy ảo Java(JVM) mà là máy ảo Dalvik (Dalvik Virtual Machine).Mỗi một máy ảo Android khi chạy đều có ít nhất một thread chính khi khởi động và có thể còn vài thread khác dùng để quản lý các tiến trình đang chạy.Ứng dụng có thể tự khởi động thêm các thread phụ vào các mục đích cụ thể.Các thread trong cùng một máy ảo tác động qua lại và được đông bộ hóa bằng cách sử dụng các đối tượng chia sẻ và các monitor liên quan đến các đối tượng đó.

Một Thread có thể là một background thread hoặc là một Foreground thread. Theo mặc định, một thread mới được tạo sẽ là foreground, Một ứng dụng chỉ kết thúc khi tất cả ForeGround Thread được kết thúc còn background thread sẽ tự động bị tiêu hủy khi ứng dụng End.

Mỗi Thread có một độ ưu tiên (số nguyên từ 1 - 10) dùng để xác định lượng thời gian CPU (CPU time ) mà thread có thể sử dụng dựa vào phương thức setPriority(int).

Phương thức của Thread trong Android cũng giống các phương thức của Thread trong Java như sleep(), wait() , notify() và notifyAll().

1. Multithreading

Multithreading có nghĩa là đa luồng, nhiều thread. Với cơ chế đa luồng, các ứng dụng của bạn có thể thực thi đồng thời nhiều dòng lệnh cùng lúc. Có nghĩa là bạn có thể làm nhiều công việc đồng thời trong cùng một ứng dụng của bạn. Có thể hiểu một cách đơn giản: hệ điều hành với cơ chế đa nhiệm cho phép nhiều ứng dụng chạy cùng lúc thì ứng dụng với cơ chế đa luồng cho phép thực hiện nhiều công việc cung lúc. Chúng ta có đã có multiprocessming nhưng cũng phải cần đến multithreading.

Vì việc tạo ra và quản lí các process cần tốn nhiều tài nguyên hệ thống hơn rất nhiều so với việc tạo ra một thread. Trong khi đó bạn có thể tạo nhiều thread song song (ít tốn tài nguyên hệ thống hơn) để thực hiện nhiều công việc cùng lúc của process. Mỗi thread thực thi một công việc hết sức đơn giản.

Khi bạn chạy một ứng dụng Android thì ứng dụng đó là một thread hoặc là nhiều thread nếu dùng multithreading.

Như đã trình bày, mỗi khi chạy ứng dụng Android thì bạn đã tạo ra thread. Dó chính là thread chính, để tạo ra một thread khác ngoài thread chính trên, chúng ta có 2 cách tương tự như Java:

* Tạo 1 lớp mới kế thừa từ lớp java.lang.Thread và viết chồng phương thức run() của lớp này
* Cung cấp 1 biến thể (instance) của thread mới với đối tượng Runnable trong quá trình khởi tạo thread này. Với cách thứ nhất, chúng ta khai báo như sau:

|  |
| --- |
| class A extends Thread  {  Public void run()  {…}  }  A a = new A();  a.start();  ……… |

Với cách này các dòng lệnh của bạn sẽ được đặt trong method run().

Với cách thứ hai, ta khai báo như sau:

|  |
| --- |
| class B extends … implements Runable  {  Public void run()  {  …//  }  }  B b = new B();  Thread t= new Thread(b);  t.start();  …….. |

Cách thứ hai vẫn phải tạo ra một đối tượng Thread

Vấn đề đặt ra là phướng thức nào tốt hơn, tối ưu hơn. Thứ nhất, Android (Java) không hỗ trợ đa kế thừa, bạn chỉ có thể extends từ một lớp duy nhất, nhưng lại có thể implements cùng lúc nhiều interface, khi ma lớp của bạn đã extends một lớp nào đó rồi thì chỉ có thể implements Runnable. Thứ hai, việc extends lớp Thread rồi dẫn tới việc bạn có thể override lại các phương thức start, stop… mà việc này có thể làm cho việc tạo thread là không thể. Vậy cách tốt nhất là implements đối tượng Runnable.

Một ví dụ đơn giản: cho một đối tượng Runnable dung để hiển thị 1 đoạn văn bản:

|  |
| --- |
| final Runnable r = new Runnable()  {  public void run()  {  tv.append("Hello World"); handler.postDelayed(this, 1000);  } };  handler.postDelayed(r, 1000); |

* **Ưu điểm của việc sử dụng Multithreading**

Mặc dù các thread chia sẻ các tài nguyên của tiến trình những vẫn có thể được thực thi 1 cách độc lập. Multi-thread hữu dụng khi dùng để lập trình đa nhiệm, khi 1 chương trình hoặc tiến trình có sự cố, toàn bộ hệ thống sẽ vẫn an toàn. Đặc biệt hữu dụng trong trường hợp một tiến trình duy nhất mà sinh ra nhiều thread con của một hệ thống đa nhiệm. Các ứng dụng Multi-thread chạy nhanh hơn trên các máy tính hỗ trợ xử lý đa luồng

* **Nhược điểm:**

Khó khăn trong việc lập trình, việc sử dụng multithreading không hề dễ dàng, chỉ lập trình viên có kinh nghiệm mới xử lí tốt trong trường hợp này. Khó khăn trong việc quản lí đồng thời, có khả năng tiềm tàng xảy ra tình trang deadlock, vì vậy cần phải nhận dạng, tránh hoặc xử lí deadlock khi nó xảy ra.

1. Sử dụng Thread trong android

Threading được sử dụng khi chương trình của bạn thực thi nhiều task tại cùng một thời điểm. Android cung cấp cho chúng ta hai phương thức sau để làm việc với Thread.

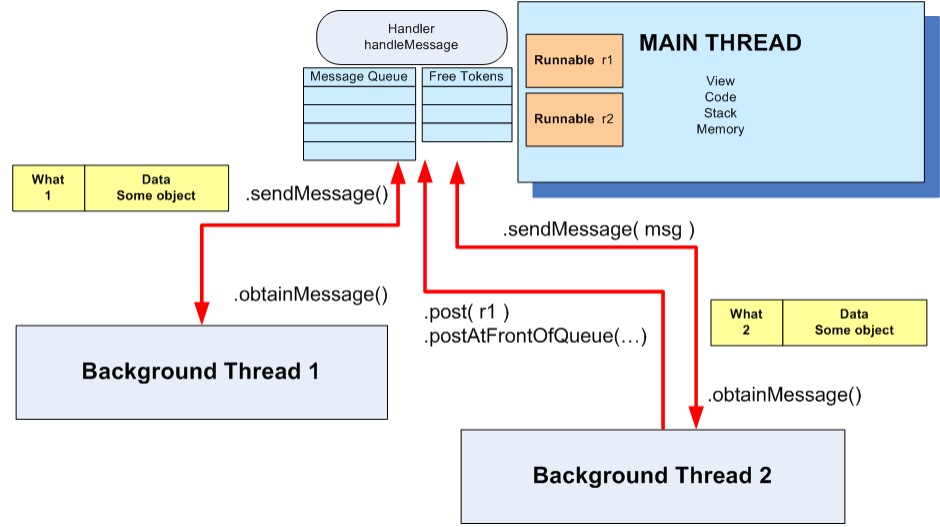
* Handler.
* AsynsTasks

1. Dùng Handler

Khi tạo một đối tượng từ class Handler, thì nó sẽ xử lý hai đối tượng đó là Messages và Runable quan hệ với Message-Queue hiện tại. Hàng đợi message chứa các công việc cần thực thi theo cơ chế FIFO. Bạn sẽ cần Handler mỗi Activity nơi mà background thread sẽ thông tin tới để update UI.

Chúng ta có thể liên lạc tới Handler nhờ hai phương thức sau:

* Messages
* Runable object



**Hình 1: Sơ đồ hoạt động Handler**

1. Sử dụng Messages

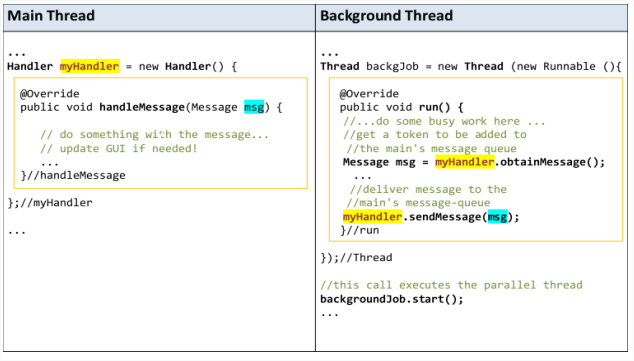
Quá trình tạo và sử dụng Handler:

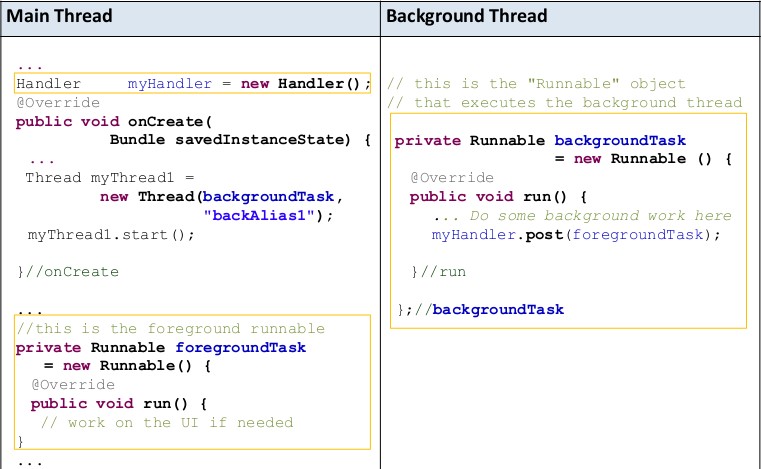
* Đầu tiền chúng ta tạo một đối tượng Handler với một quan hệ với một phương thức callback để xử lý các messages nhận về.
* Tiếp đó từ background thread chúng sẽ cần send một messages tới handler
* Để gởi 1 Message tới Handler Thread phải dẫn tới obtainMessage() để lấy object Message ra.
* Có vài cách để sử dụng obtainMessage(). Có thể truyền rỗng hoặc truyền có đối số.

|  |
| --- |
| Vd: // thread 1 produces some local data  String localData = “Greeting from thread 1”;  // thread 1 requests a message & adds localData to it  Message mgs = myHandler.obtainMessage (1, localData); |

* Khi gởi 1 Message tới Handler Thread có thể dùng 1 trong các phương thức:

|  |
| --- |
| sendMessage()  sendMessageAtFrontOfQueue()  sendMessageAtTime()  sendMessageDelayed() |



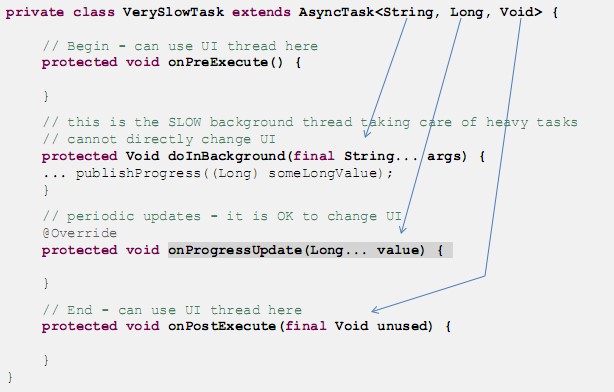
1. Sử dụng Post Method

1. Dùng AsyncTask

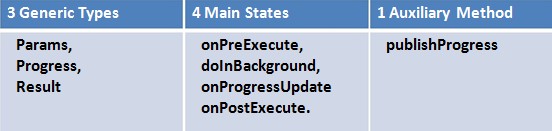
Khi ứng dụng của bạn thực hiện 1 task vụ dài, chiếm 1 khoảng thời gian nhất định. Ví dụ khi click nút Login, ứng dụng phải gửi request lên server để xử lý và trả kết quả về. Nếu làm theo kiểu bình thường thì ứng dụng của bạn sẽ có cảm giác bị treo, bị đứng hình. Do đó chúng ta có thể sử dụng **AsyncTask** trong tình huống này để giải quyết vấn đề trên.

AsyncTask cho phép bạn thực hiện hành động xử lý phía background, và trả kết quả lên UI thread mà không cần phải xử lý Thread/ Handler.

Khái báo một lớp và kế thừa lại lớp AsyncTask như sau:



* AsyncTask cho phép sử dụng dễ dàng các UI Thread
* Lớp này cho phép thực hiện trên các hoạt động nền và đưa ra kết quả trên UI Thread mà không cần phải thao tác với Thread hoặc handler.
* Một AsyncTask được định nghĩa bởi một phép tính toán chạy trên background thread và kết quả được công bố trên UI Thread
* Một AsyncTask được định nghĩa bởi:

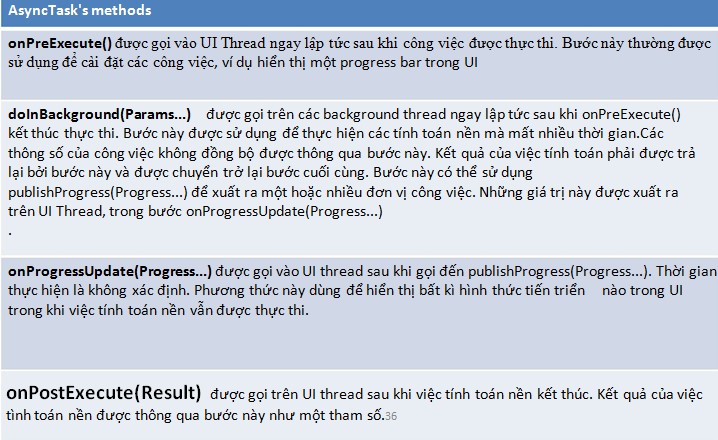


AsyncTask <Params, Progress, Result>.Trong đó:

* **Params**: các tham số được gửi đến nhiệm vụ khi thực thi.
* **Progress:** các xử lý sẽ được thực hiện trong AsynTask.
* **Result**: Kết quả xử lý.

Không phải tất cả các kiểu luôn được sử dụng bởi AsyncTask. Để không sử dụng một loại tham số thì chỉ cần sử dụng kiểu Void.

**Chú ý:** Cú pháp "String ..." là mảng các giá trị String, tương tự như "String []"



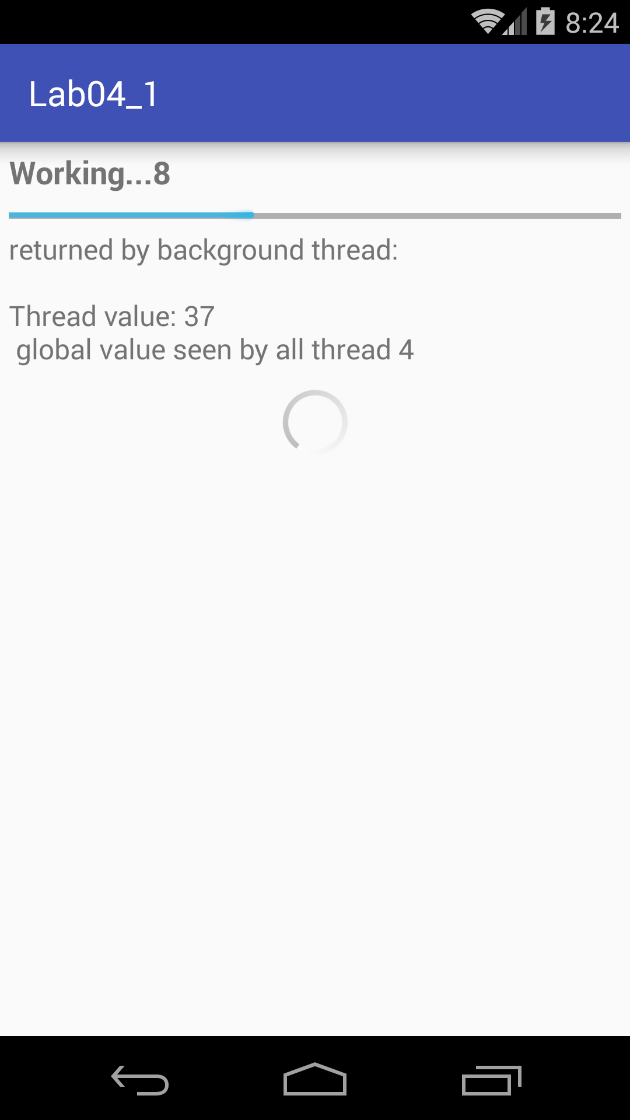
Nhiệm vụ chính gọi AsyncTask để làm một số công việc chậm. Các phương pháp AsyncTask làm yêu cầu tính toán và định kỳ cập nhật giao diện người dùng chính.

**Lưu ý**

1-Instance của AsyncTask chỉ được dùng trong UI thread.

2-Method onPostExecute, onPreExecute, onProgressUpdate là optional

3-Task execute chỉ được gọi 1 lần thôi, gọi nhiều hơn sẽ văng lỗi

1. THỰC HÀNH
2. Viết một ứng dụng MultiThread sử dụng Message. Giao diện:
3. Mô tả ứng dụng
   * Ứng dụng có một button Start. Khi user nhấn vào Start, thread mới sẽ khởi chạy.
   * Trong thread mới sẽ phát sinh ngẫu nhiên một số từ 0 đến 100 đồng thời tăng biến đếm global intTest lên một giá trị.
   * Thread sẽ gửi message chứa các dữ liệu trên. UI thread sẽ sử dụng Handler để nhận message và cập nhật giao diện.
4. Các bước thực hiện

* Xây dựng các file dimens.xml

|  |  |
| --- | --- |
| **Name** | **Value** |
| margin\_base | 5dp |
| text\_small | 14sp |
| text\_medium | 16sp |
| text\_medium\_large | 18sp |
| text\_large | 20sp |

* Xây dựng file strings.xml

|  |  |
| --- | --- |
| **Name** | **Value** |
| **start** | Start |
| **returned\_by\_bg\_thread** | Returned by background thread: \n\n |
| **done\_background\_thread\_has\_been\_stopped** | Done \nBackground thread has been stopped |
| **done** | Done |
| **working** | Working... |
| **global\_value\_seen** | \n global value seen by all thread |

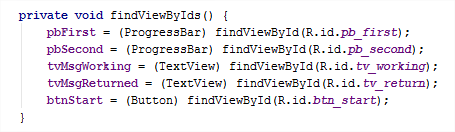
* Xây dựng giao diện

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"  android:padding="@dimen/margin\_base"  tools:context="uit.edu.vn.lab04\_1.MainActivity"**>   <**TextView  android:id="@+id/tv\_working"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:textSize="@dimen/text\_medium\_large"  android:textStyle="bold"** />   <**ProgressBar  android:id="@+id/pb\_first"  style="?android:attr/progressBarStyleHorizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_marginTop="@dimen/margin\_base"** />   <**TextView  android:id="@+id/tv\_return"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:textSize="@dimen/text\_medium"** />   <**FrameLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"**>   <**Button  android:id="@+id/btn\_start"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="center"  android:text="@string/start"** />   <**ProgressBar  android:id="@+id/pb\_second"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_gravity="center"  android:layout\_marginTop="@dimen/margin\_base"** />  </**FrameLayout**> </**LinearLayout**> |

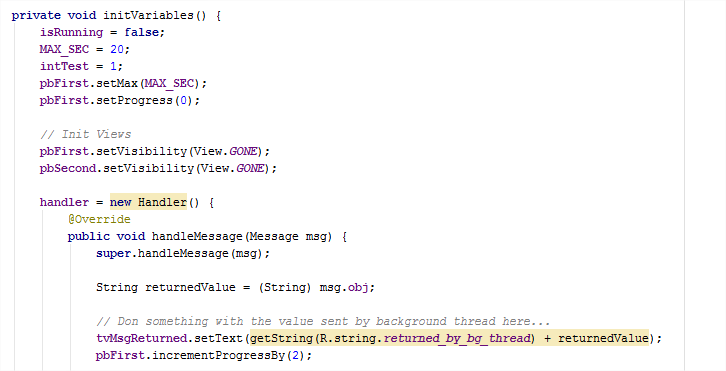
* Khai báo biến

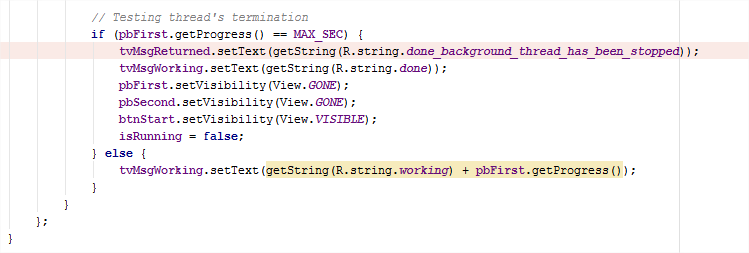
|  |
| --- |
| **private** ProgressBar **pbFirst**, **pbSecond**; **private** TextView **tvMsgWorking**, **tvMsgReturned**; **private boolean isRunning**; **private int MAX\_SEC**; **private int intTest**; **private** Thread **bgThread**; **private** Handler **handler**; **private** Button **btnStart**; |

* Xây dựng hàm findViewByIds

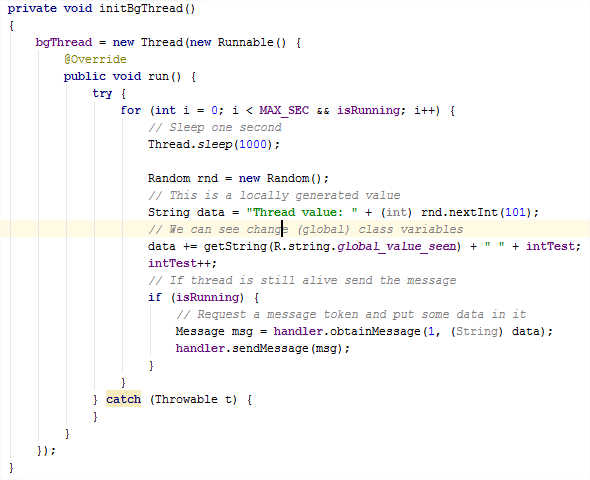


* Xây dựng hàm initVảiables



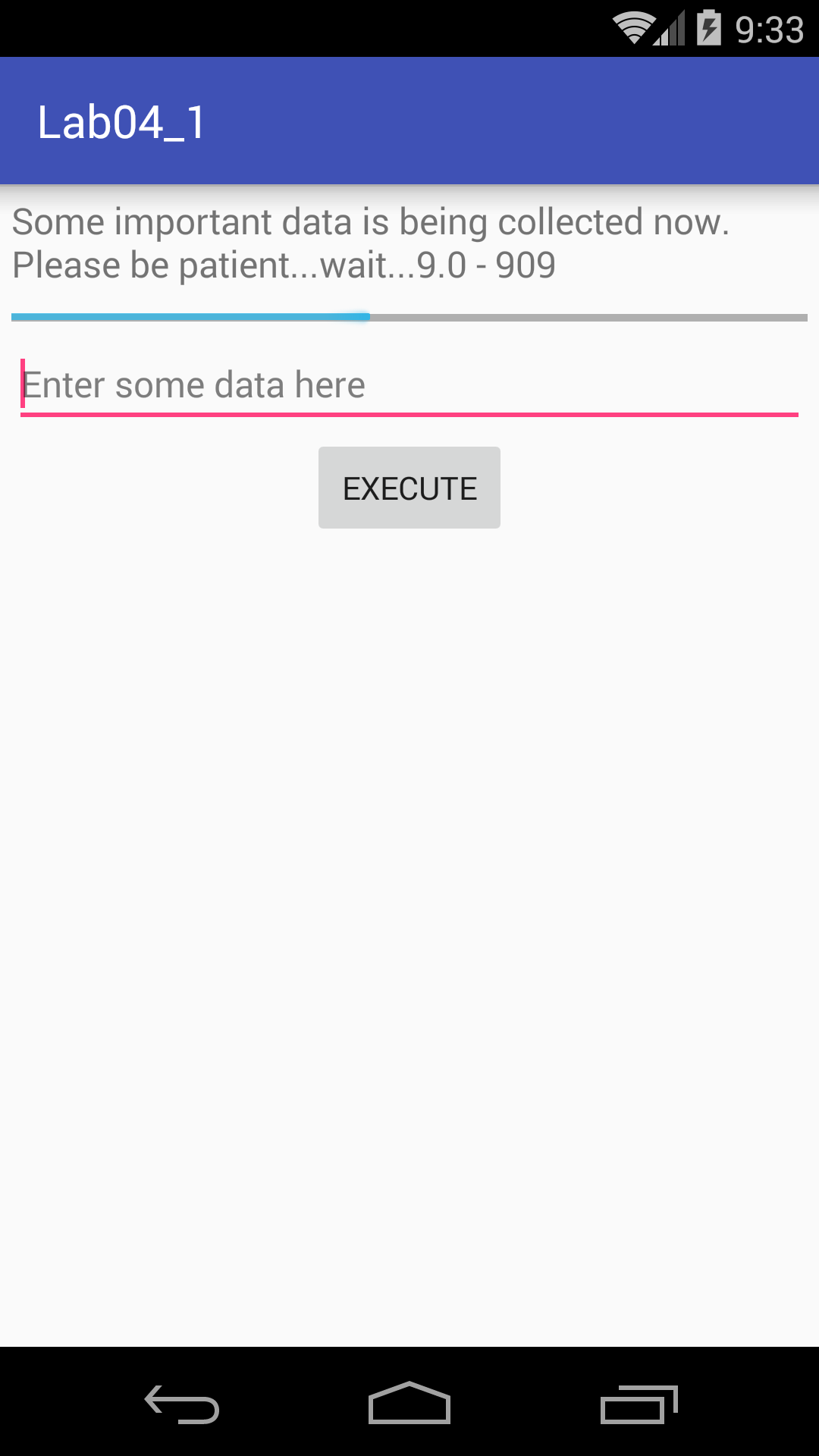


* Xây dựng hàm initBgThread



* Override các hàm onStart, onStop, onCreate

|  |
| --- |
|  |
| @Override **protected void** onStart() {  **super**.onStart();   initBgThread(); } |
| @Override **protected void** onStop() {  **super**.onStop();  **isRunning** = **false**; } |

1. Viết ứng dụng MultiThread sử dụng Post
2. Mô tả ứng dụng
   * Ứng dụng khởi chạy 1 thread mới. Thread mới sẽ chạy 2 runnable: bgRunnable và fgRunnable cùng cập nhật giá trị của biến globalValue. Trong đó, bgRunnable cứ sau 1s sẽ tăng giá trị biến globalValue lên 1. Còn fgRunnable mỗi lần sẽ tăng giá trị biến globalValue lên 100.
   * Khi người dùng nhập data vào etInput và nhấn Execute thì data sẽ được hiện lên dưới dạng Toast.
3. Các bước thực hiện

* Xây dựng các file dimens.xml

|  |  |
| --- | --- |
| **Name** | **Value** |
| margin\_base | 5dp |
| text\_small | 14sp |
| text\_medium | 16sp |
| text\_medium\_large | 18sp |
| text\_large | 20sp |

* Xây dựng file strings.xml

|  |  |
| --- | --- |
| **Name** | **Value** |
| **bg\_work\_is\_over** | Background work is over! |
| **execute** | Execute |
| **enter\_some\_data\_here** | Enter some data here |

* Xây dựng giao diện

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"  android:padding="@dimen/margin\_base"  tools:context="uit.edu.vn.lab04\_1.MainActivity"**>   <**TextView  android:id="@+id/tv\_top\_caption"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:textSize="@dimen/text\_medium"** />   <**ProgressBar  android:id="@+id/pb\_waiting"  style="?android:attr/progressBarStyleHorizontal"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_marginTop="@dimen/margin\_base"** />   <**EditText  android:id="@+id/et\_input"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:hint="@string/enter\_some\_data\_here"  android:textSize="@dimen/text\_medium"** />   <**Button  android:id="@+id/btn\_execute"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_gravity="center"  android:text="@string/execute"** />  </**LinearLayout**> |

* Khai báo biến

|  |
| --- |
| **private** ProgressBar **pbWaiting**; **private** TextView **tvTopCaption**; **private** EditText **etInput**; **private** Button **btnExecute**; **private int globalValue**, **accum**; **private long startTime**; **private final** String **PATIENCE** = **"Some important data is being collected now.\nPlease be patient...wait..."**; **private** Handler **handler**; **private** Runnable **fgRunnable**, **bgRunnable**; **private** Thread **testThread**; |

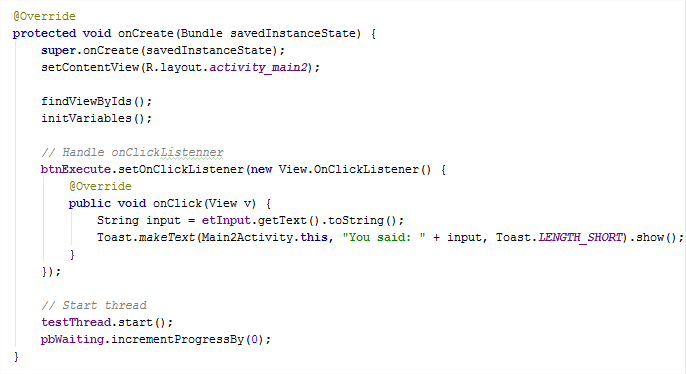
* Xây dựng hàm findViewByIds

|  |
| --- |
| **tvTopCaption** = (TextView) findViewById(R.id.***tv\_top\_caption***); **pbWaiting** = (ProgressBar) findViewById(R.id.***pb\_waiting***); **etInput** = (EditText) findViewById(R.id.***et\_input***); **btnExecute** = (Button) findViewById(R.id.***btn\_execute***); |

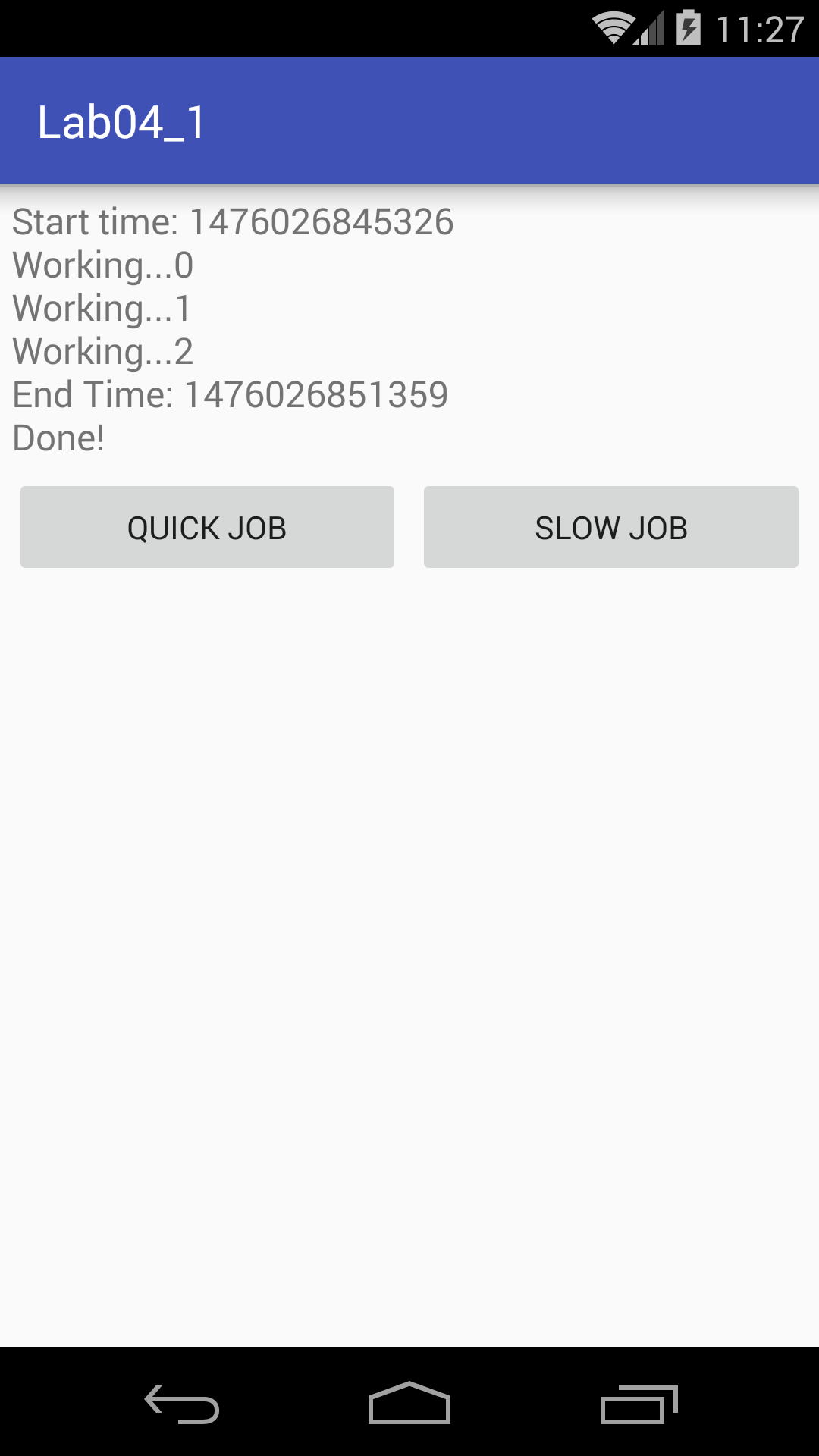
* Xây dựng hàm initVariables

|  |
| --- |
| **globalValue** = 0; **accum** = 0; **startTime** = System.*currentTimeMillis*(); **handler** = **new** Handler(); |

* Override hàm onCreate



Chú ý: Khi nhiều thread cùng tương tác với một đối tượng, bạn cần phải điều khiển chúng một cách cẩn thận để tránh tranh chấp tài nguyên. Việc bổ sung từ khóa synchronized vào khai báo phương thức nhằm đảm bảo chỉ có một thread được phép ở bên trong phương thức tại một thời điểm.

1. Viết ứng dụng sử dụng Asynctask
2. Mô tả ứng dụng
   * Ứng dụng gồm 2 button Quick Job và Slow Job.
   * Khi user nhấn vào Quick Job, thời gian hiện hành được hiển thị ngay lên tvStatus.
   * Khi user nhấn vào Slow Job, ứng dụng sẽ sử dụng AsynTask để thực hiện tăng giá trị biến i sau mỗi 2s. Giá trị biến i được cập nhật lên giao diện thông qua hàm publishProgress và onProgressUpdate. Trong quá trình chạy AsynTask, dialog sẽ được hiển thị lên trong thời gian chờ.
3. Các bước thực hiện

* Xây dựng các file dimens.xml

|  |  |
| --- | --- |
| **Name** | **Value** |
| margin\_base | 5dp |
| text\_small | 14sp |
| text\_medium | 16sp |
| text\_medium\_large | 18sp |
| text\_large | 20sp |

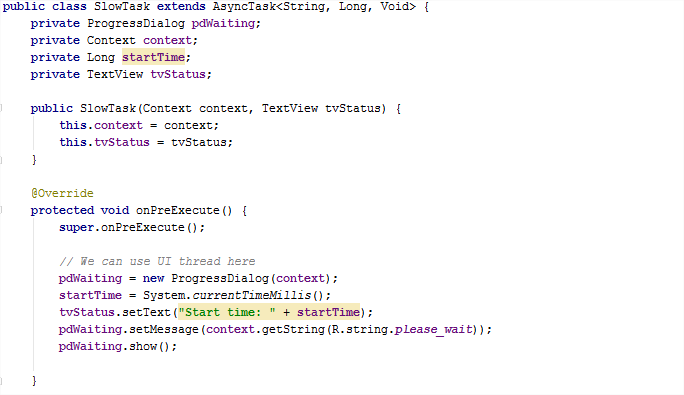
* Xây dựng file strings.xml

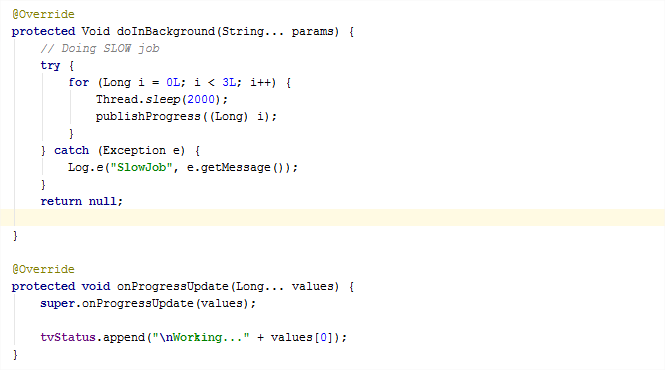
|  |  |
| --- | --- |
| **Name** | **Value** |
| **quick\_job** | Quick Job |
| **slow\_job** | Slow Job |
| **please\_wait** | Some SLOW job is being done. Please wait... |

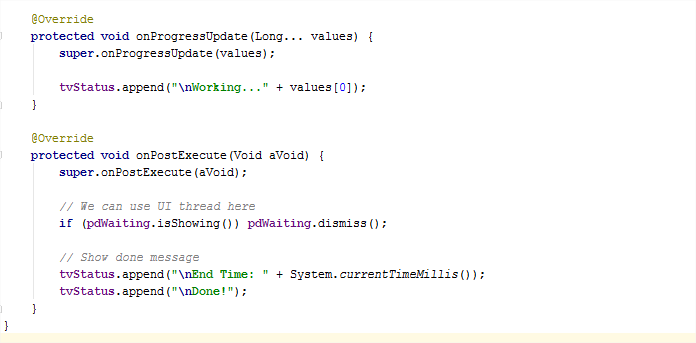
* Xây dựng giao diện

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>* <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:orientation="vertical"  android:padding="@dimen/margin\_base"  tools:context="uit.edu.vn.lab04\_1.Main3Activity"**>   <**TextView  android:id="@+id/tv\_status"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:textSize="@dimen/text\_medium"** />   <**LinearLayout  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_marginTop="@dimen/margin\_base"  android:orientation="horizontal"**>   <**Button  android:id="@+id/btn\_quick\_job"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="@string/quick\_job"** />   <**Button  android:id="@+id/btn\_slow\_job"  android:layout\_width="0dp"  android:layout\_height="wrap\_content"  android:layout\_weight="1"  android:text="@string/slow\_job"  android:layout\_marginLeft="@dimen/margin\_base"**/>  </**LinearLayout**>  </**LinearLayout**> |

* Xây dựng class SlowTask







* Xử lý trong MainActivity

|  |
| --- |
| **public class** Main3Activity **extends** AppCompatActivity {   **private** Button **btnQuickJob**, **btnSlowJob**;  **private** TextView **tvStatus**;  **private** SlowTask **slowTask**;   @Override  **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_main3***);   findViewByIds();   *// Init slowtask* **slowTask**=**new** SlowTask(Main3Activity.**this**, **tvStatus**);   *// Handle onClickListenner* **btnQuickJob**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  SimpleDateFormat sdf=**new** SimpleDateFormat(**"dd/MM/yyyy HH:mm:ss"**);  **tvStatus**.setText(sdf.format(**new** Date()));  }  });  **btnSlowJob**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  **slowTask**.execute();  }  });  }   **private void** findViewByIds() {  **btnQuickJob** = (Button) findViewById(R.id.***btn\_quick\_job***);  **btnSlowJob** = (Button) findViewById(R.id.***btn\_slow\_job***);  **tvStatus** = (TextView) findViewById(R.id.***tv\_status***);  } } |

1. THAM KHẢO THÊM (Tự xem thêm)
2. **Công cụ code nhóm (đồng bộ code)**
   * Source code hosting:

+ Github: <https://github.com/>

+ Bitbucket: <https://bitbucket.org>

* + Tools:

+ Github: <https://desktop.github.com/>

+ TortoiseGit: <https://tortoisegit.org/>

+ SourceTree: <https://www.sourcetreeapp.com/>

* Các thao tác: tạo, clone repository; pull, commit, push, fix conflict; sử dụng branch.

1. Cách import library vào project:

Tìm hiểu cách Import/export library trong các trường hợp cụ thể:

* + Tạo 1 project library, xong include vô project main.
  + Tạo 1 project library, build file \*.aar, xong include file \*.aar vô project main.
  + Include 3rd libs bằng file \*.jar hoặc \*.aar.
  + Get source 3rd libs, include vô project main.
  + Get source 3rd libs, build file \*.aar, rồi include file \*.aar vô project main.

1. **Calligraphy**
   * <https://github.com/chrisjenx/Calligraphy>
2. Drawable
   * <https://developer.android.com/guide/topics/resources/drawable-resource.html>
   * <http://www.vogella.com/tutorials/AndroidDrawables/article.html>