MultiThreading

PHÁT TRIỂN ỨNG DỤNG TRÊN THIẾT BỊ DI ĐỘNG

LAB 05

I. Thread và Multithreading

1. Thread

Thread là đơn vị nhỏ nhất của tiến trình được bao hàm trong các tiến trình thực thi của máy tính. Máy ảo Java (JVM) cho phép một ứng dụng có thể chạy nhiều thread trong thời điểm thực thi.

Android chạy trên một hệ điều hành nhân Linux. Mỗi một máy ảo Android đều có ít nhất một thread chính khi khởi động và có thể chạy thêm các thread khác để quản lý các tiến trình đang hoạt động. 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().

2. 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 mà 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.

II. 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.

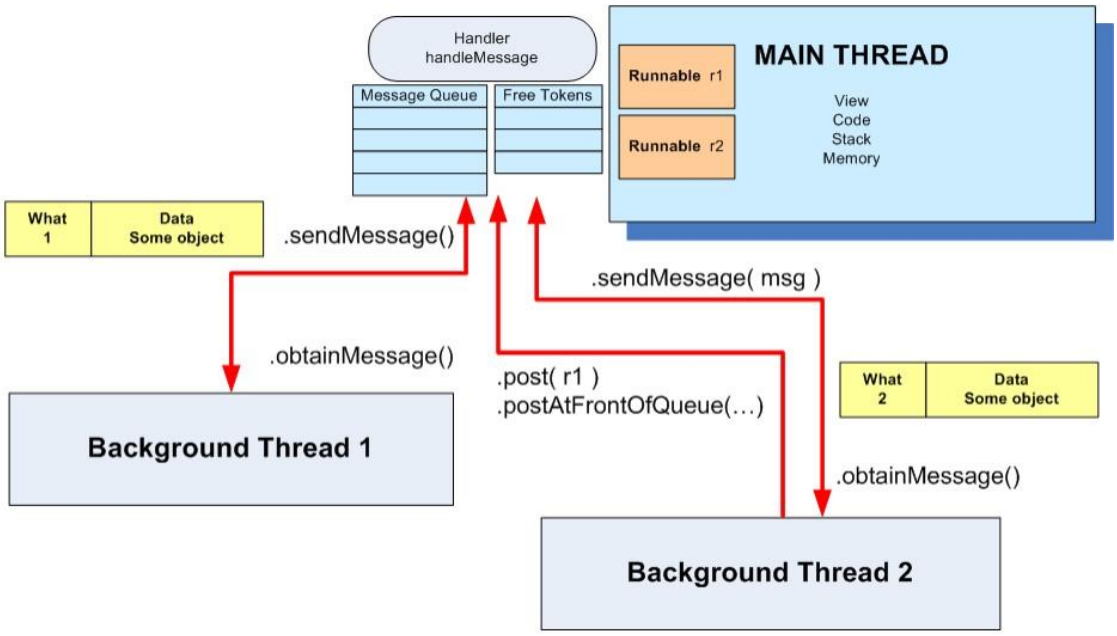
- Handler.
- AsynsTasks

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

a) 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()
```

Main Thread

```
...
Handler myHandler = new Handler() {

  @Override
  public void handleMessage(Message msg) {

     // do something with the message...
     // update GUI if needed!
     ...
  }//handleMessage

};//myHandler

...
```

Background Thread

```
...
Thread backgJob = new Thread (new Runnable (){

  @Override
  public void run() {
   //...do some busy work here ...
   //get a token to be added to
    //the main's message queue
   Message msg = myHandler.obtainMessage();
     ...
    //deliver message to the
   //main's message-queue
   myHandler.sendMessage(msg);
  }//run

});//Thread

//this call executes the parallel thread
backgroundJob.start();
...
```

b) Sử dụng Post Method

Main Thread

```
...
Handler     myHandler = new Handler();
@Override
public void onCreate(
          Bundle savedInstanceState) {
 ...
 Thread myThread1 =
        new Thread(backgroundTask,
                   "backAlias1");
 myThread1.start();

}//onCreate

...
//this is the foreground runnable
private Runnable foregroundTask
   = new Runnable() {
  @Override
  public void run() {
    // work on the UI if needed
}
...
```

Background Thread

```
// this is the "Runnable" object
// that executes the background thread
private Runnable backgroundTask
                = new Runnable () {
  @Override
  public void run() {
     ... Do some background work here
     myHandler.post(foregroundTask);

  }//run

};//backgroundTask
```

2. Dùng AsyncTask (Hiện không còn được khuyến khích sử dụng)

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:

```
private class VerySlowTask extends AsyncTask<String, Long, Void> {

    // Begin - can use UI thread here
    protected void onPreExecute() {

    }

    // this is the SLOW background thread taking care of heavy tasks
    // cannot directly change UI
    protected Void doInBackground(final String... args) {
    ... publishProgress((Long) someLongValue);
    }

    // periodic updates - it is OK to change UI
    @Override
    protected void onProgressUpdate(Long... value) {

    }

    // End - can use UI thread here
    protected void onPostExecute(final Void unused) {

    }
}
```

- 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:

3 Generic Types	4 Main States	1 Auxiliary Method
Params, Progress, Result	onPreExecute, doInBackground, onProgressUpdate onPostExecute.	publishProgress

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 []"

AsyncTask's methods
onPreExecute() được gọi vào UI Thread ngay lập tức sau khi công việc được thực thi. Bước này thường được sử dụng để cài đặt các công việc, ví dụ hiển thị một progress bar trong UI
doInBackground(Params...) được gọi trên các background thread ngay lập tức sau khi onPreExecute() kết thúc thực thi. Bước này được sử dụng để thực hiện các tính toán nền mà mất nhiều thời gian.Các thông số của công việc không đồng bộ được thông qua bước này. Kết quả của việc tính toán phải được trả lại bởi bước này và được chuyển trở lại bước cuối cùng. Bước này có thể sử dụng publishProgress(Progress...) để xuất ra một hoặc nhiều đơn vị công việc. Những giá trị này được xuất ra trên UI Thread, trong bước onProgressUpdate(Progress...) .
onProgressUpdate(Progress...) được gọi vào UI thread sau khi gọi đến publishProgress(Progress...). Thời gian thực hiện là không xác định. Phương thức này dùng để hiển thị bất kì hình thức tiến triển nào trong UI trong khi việc tính toán nền vẫn được thực thi.
onPostExecute(Result) được gọi trên UI thread sau khi việc tính toán nền kết thúc. Kết quả của việc tình toán nền được thông qua bước này như một tham số.36

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

3. RxJava, Rx Android

- **RxJava** là một trong những Reactive Extension, dành cho ngôn ngữ Java. Về cơ bản nó là một thư viện follow theo phong cách Observer Pattern.

Chúng ta có thể tạo ra bất kì luồng dữ liệu không đồng bộ trên bất kỳ thread nào, chuyển đổi dữ liệu và dữ liệu này được sử dụng bởi Observer trên bất kỳ thread nào.

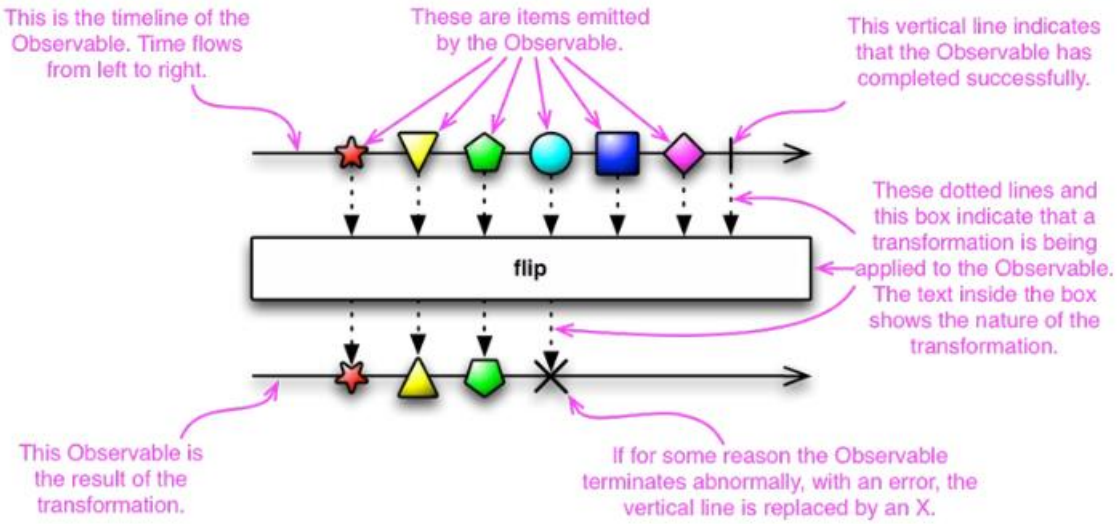
Thư viện này cung cấp nhiều toán tử tuyệt vời như Map, Combine, Merge, Filter, ... có thể áp dụng cho một luồng dữ liệu.

- **RxAndroid** là một loại Rx dành cho nền tảng Android. Nó được hình thành từ RxJava với vài lớp được thêm vào, giúp áp dụng reactive programming trong Android dễ dàng, hiệu quả hơn.

Ví dụ: nếu một task chạy quá lâu trên main thread sẽ gây ra **lỗi ANR**, và muốn update giao diện bạn phải update trên main thread.

==> **RxAndroid** giới thiệu thêm chức năng cho lớp **Schedulers**, giúp update giao diện trên **mainthread, từ một Looper.**

Các Thành phần cơ bản khi sử dụng RxJava, RxAndroid



Observable

Là một luồng dữ liệu (data stream) làm công việc nào đó và phát ra dữ liệu (data). Theo hình vẽ thì đó là đường mũi tên ở trên, thực hiện xử lý gì đó và phát ra các dữ liệu là các khối hình.

Observer

Là những đối tượng lắng nghe Observable. Nó nhận dữ liệu từ Observable phát ra. Theo hình vẽ thì nó là đường mũi tên ở dưới, nhận dữ liệu từ Observer là các khối hình sau khi đã được xoay.

Subscription

Sự liên kết giữa Observable và Observer được gọi là Subscription. Có thể có nhiều Observers đăng ký (subscribed) tới chỉ một Observable.

Schedulers

Schedulers quyết định thread mà trên đó Observable sẽ phát ra dữ liệu và trên Observer sẽ nhận được dữ liệu

Operator / Transformation

Operator còn có thể gọi là Transformation bởi vì nó là các toán tử có nhiệm vụ biến đổi dữ liệu được phát ra bởi Observable trước khi một Observer nhận chúng (nhận dữ liệu)

Trong RxJava, bạn có thể tạo một Observable từ các nguồn dữ liệu khác nhau như danh sách, tập hợp, sự kiện giao diện người dùng, kết quả truy vấn cơ sở dữ liệu, gọi API mạng, và nhiều nguồn dữ liệu khác.

Chúng ta sẽ có 5 loại Observable sau:

- **Observable**
- **Single**
- **Maybe**
- **Flowable**
- **Completable**

Để tạo Observable trong RxJava, bạn có thể sử dụng các phương thức như:

- **Observable.create()**: Tạo một Observable từ mã logic tùy chỉnh. Bạn có thể sử dụng các phương thức của Observer để phát ra các sự kiện hoặc giá trị dữ liệu.

- **Observable.just()**: Tạo một Observable từ một hoặc nhiều giá trị cụ thể. Observable này sẽ phát ra các giá trị này và hoàn thành sau đó.
- **Observable.interval()**: Tạo một Observable phát ra các số nguyên liên tục sau một khoảng thời gian nhất định.
- **Observable.fromIterable()**: Tạo một Observable từ một danh sách, một tập hợp hoặc một iterable.
- **Observable.fromCallable()**: Tạo một Observable từ một Callable, nơi bạn có thể thực hiện các tác vụ bất đồng bộ và trả về một giá trị

Trong RxJava, có thể tạo một Observer bằng cách triển khai đối tượng Observer<T>. Đối tượng này định nghĩa các phương thức mà bạn cần triển khai để xử lý các sự kiện và giá trị từ Observable.

Các phương thức chính trong giao diện Observer bao gồm:

- **onNext(T value)**: Phương thức này được gọi khi một giá trị mới được phát ra từ Observable. Bạn có thể định nghĩa các hành động xử lý khi nhận được giá trị này.
- **onError(Throwable throwable)**: Phương thức này được gọi khi có một lỗi xảy ra trong quá trình phát ra giá trị từ Observable. Bạn có thể xử lý và báo cáo lỗi trong phương thức này.
- **onComplete()**: Phương thức này được gọi khi Observable hoàn thành việc phát ra các giá trị. Bạn có thể thực hiện các hành động dọn dẹp hoặc xử lý cuối cùng trong phương thức này.

Ví dụ sử dụng **RxJava** thay thế cho **Asynctask**

```
import android.os.Bundle;
import android.util.Log;

import androidx.appcompat.app.AppCompatActivity;

import io.reactivex.rxjava3.android.schedulers.AndroidSchedulers;
import io.reactivex.rxjava3.core.Observable;
import io.reactivex.rxjava3.core.Observer;
import io.reactivex.rxjava3.disposables.Disposable;
import io.reactivex.rxjava3.schedulers.Schedulers;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = MainActivity.class.getSimpleName();
```

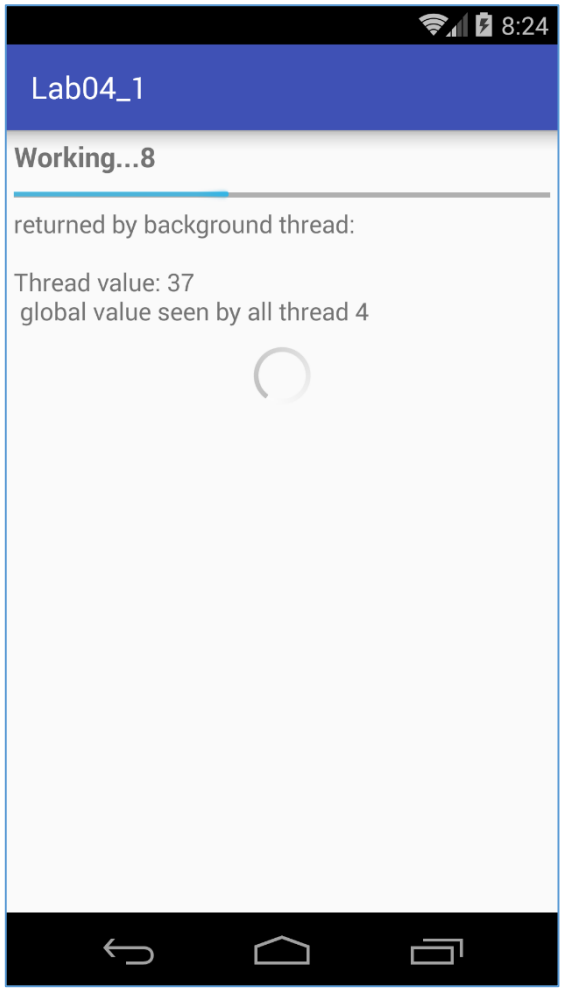
```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Thực hiện tác vụ không đồng bộ sử dụng RxJava
        performTask();
    }

    private void performTask() {
        Observable.fromCallable(this::doBackgroundTask)
                .subscribeOn(Schedulers.io()) // Thực hiện tác vụ trên luồng I/O
                .observeOn(AndroidSchedulers.mainThread()) // Nhận kết quả trên luồng chính (UI thread)
                .subscribe(new Observer<String>() {
                    @Override
                    public void onSubscribe(Disposable d) {
                        // Không cần thực hiện gì khi đã subscribe
                    }

                    @Override
                    public void onNext(String result) {
                        // Xử lý kết quả trả về từ tác vụ
                        Log.d(TAG, "onNext: " + result);
                    }

                    @Override
                    public void onError(Throwable e) {
                        // Xử lý khi có lỗi xảy ra
                        Log.e(TAG, "onError: ", e);
                    }

                    @Override
                    public void onComplete() {
                        // Xử lý khi tác vụ hoàn thành
                    }
                });
    }
```

```
    private String doBackgroundTask() {
        // Mô phỏng một tác vụ không đồng bộ, ví dụ: lấy dữ liệu từ máy chủ
        try {
            Thread.sleep(3000); // Mô phỏng tác vụ mất thời gian
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return "Task completed!";
    }
}
```

Trong ví dụ này, chúng ta sử dụng RxJava để thực hiện một tác vụ không đồng bộ (`doBackgroundTask()`) trên một luồng I/O (`Schedulers.io()`) và nhận kết quả trên luồng chính (UI thread) (`AndroidSchedulers.mainThread()`). Kết quả của tác vụ được trả về thông qua một Observer.

III. THỰC HÀNH

1. Viết một ứng dụng MultiThread sử dụng Message. Giao diện:



a) 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.

b) 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

```
private void findViewByIds() {
    pbFirst = (ProgressBar) findViewById(R.id.pb_first);
    pbSecond = (ProgressBar) findViewById(R.id.pb_second);
    tvMsgWorking = (TextView) findViewById(R.id.tv_working);
    tvMsgReturned = (TextView) findViewById(R.id.tv_return);
    btnStart = (Button) findViewById(R.id.btn_start);
}
```

- Xây dựng hàm initVảiables

```
private void initVariables() {
    isRunning = false;
    MAX_SEC = 20;
    intTest = 1;
    pbFirst.setMax(MAX_SEC);
    pbFirst.setProgress(0);

    // Init Views
    pbFirst.setVisibility(View.GONE);
    pbSecond.setVisibility(View.GONE);

    handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);

            String returnedValue = (String) msg.obj;

            // Don something with the value sent by background thread here...
            tvMsgReturned.setText(getString(R.string.returned_by_bg_thread) + returnedValue);
            pbFirst.incrementProgressBy(2);

            // Testing thread's termination
            if (pbFirst.getProgress() == MAX_SEC) {
                tvMsgReturned.setText(getString(R.string.done_background_thread_has_been_stopped));
                tvMsgWorking.setText(getString(R.string.done));
                pbFirst.setVisibility(View.GONE);
                pbSecond.setVisibility(View.GONE);
                btnStart.setVisibility(View.VISIBLE);
                isRunning = false;
            } else {
                tvMsgWorking.setText(getString(R.string.working) + pbFirst.getProgress());
            }
        }
    };
}
```

- Xây dựng hàm initBgThread

```
private void initBgThread()
{
    bgThread = new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                for (int i = 0; i < MAX_SEC && isRunning; i++) {
                    // Sleep one second
                    Thread.sleep(1000);

                    Random rnd = new Random();
                    // This is a locally generated value
                    String data = "Thread value: " + (int) rnd.nextInt(101);
                    // We can see change (global) class variables
                    data += getString(R.string.global_value_seen) + " " + intTest;
                    intTest++;
                    // If thread is still alive send the message
                    if (isRunning) {
                        // Request a message token and put some data in it
                        Message msg = handler.obtainMessage(1, (String) data);
                        handler.sendMessage(msg);
                    }
                }
            } catch (Throwable t) {
            }
        }
    });
}
```

- Override các hàm onStart, onStop, onCreate

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

    findViewByIds();
    initVariables();

    // Handle clickListenner
    btnStart.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            isRunning = true;
            pbFirst.setVisibility(View.VISIBLE);
            pbSecond.setVisibility(View.VISIBLE);
            btnStart.setVisibility(View.GONE);
            bgThread.start();
        }
    });
}
```

```
@Override
protected void onStart() {
    super.onStart();

    initBgThread();
}
```

```
@Override
protected void onStop() {
    super.onStop();
    isRunning = false;
}
```

2. Viết ứng dụng MultiThread sử dụng Post

a) 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.

b) Các bước thực hiện

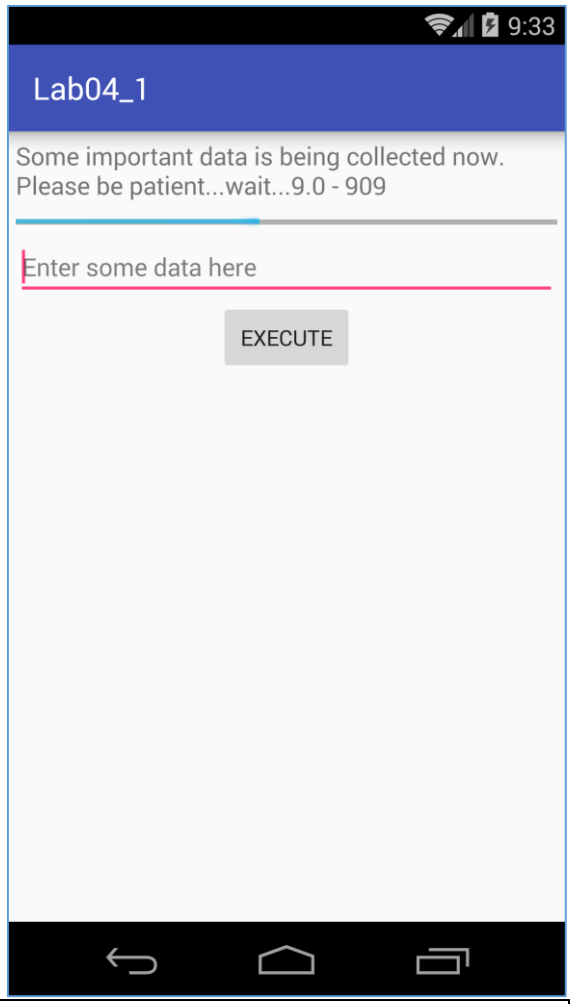
- Xây dựng các file dimens.xml

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();
```

```
fgRunnable = new Runnable() {
    @Override
    public void run() {
        try {
            // Calculate new value
            int progressStep = 5;
            double totalTime = (System.currentTimeMillis() - startTime) / 1000;
            synchronized (this) {
                globalValue += 100;
            }

            // Update UI
            tvTopCaption.setText(PATIENCE + totalTime + " - " + globalValue);
            pbWaiting.incrementProgressBy(progressStep);
            accum += progressStep;

            // Check to stop
            if (accum >= pbWaiting.getMax()) {
                tvTopCaption.setText(getString(R.string.bg_work_is_over));
                pbWaiting.setVisibility(View.GONE);
            }
        } catch (Exception e) {
            Log.e("fgRunnable", e.getMessage());
        }
    }
};
```

```
bgRunnable = new Runnable() {
    @Override
    public void run() {
        try {
            for (int i = 0; i < 20; i++) {
                // Sleep 1 second
                Thread.sleep(1000);

                // Now talk to main thread
                // Optinally change some global variable such as: globalValue

                synchronized (this) {
                    globalValue += 1;
                }

                handler.post(fgRunnable);
            }
        } catch (Exception e) {
            Log.e("bgRunnable", e.getMessage());
        }
    }
};

testThread = new Thread(bgRunnable);
```

- Override hàm onCreate

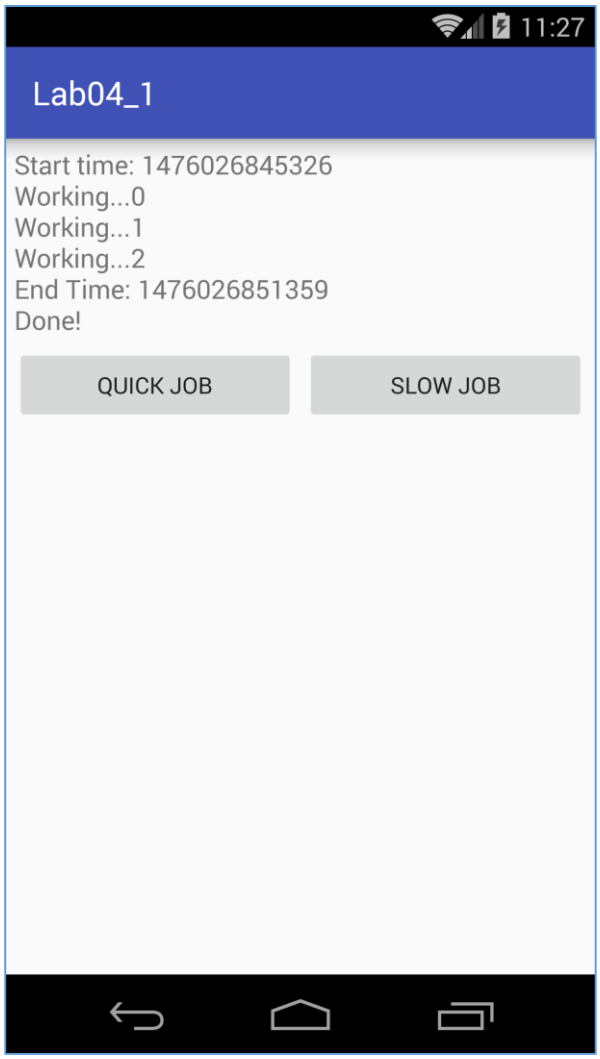
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);

    findViewByIds();
    initVariables();

    // Handle onClickListenner
    btnExecute.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String input = etInput.getText().toString();
            Toast.makeText(Main2Activity.this, "You said: " + input, Toast.LENGTH_SHORT).show();
        }
    });

    // Start thread
    testThread.start();
    pbWaiting.incrementProgressBy(0);
}
```

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.

3. Viết ứng dụng sử dụng Asynctask (Hiện nay không khuyến khích dùng nữa nhưng Cô vẫn để vào để các biết)

a) 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.



b) 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

```
public class SlowTask extends AsyncTask<String, Long, Void> {
    private ProgressDialog pdWaiting;
    private Context context;
    private Long startTime;
    private TextView tvStatus;

    public SlowTask(Context context, TextView tvStatus) {
        this.context = context;
        this.tvStatus = tvStatus;
    }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();

        // We can use UI thread here
        pdWaiting = new ProgressDialog(context);
        startTime = System.currentTimeMillis();
        tvStatus.setText("Start time: " + startTime);
        pdWaiting.setMessage(context.getString(R.string.please_wait));
        pdWaiting.show();

    }
```

```
@Override
protected Void doInBackground(String... params) {
    // Doing SLOW job
    try {
        for (Long i = 0L; i < 3L; i++) {
            Thread.sleep(2000);
            publishProgress((Long) i);
        }
    } catch (Exception e) {
        Log.e("SlowJob", e.getMessage());
    }
    return null;

}

@Override
protected void onProgressUpdate(Long... values) {
    super.onProgressUpdate(values);

    tvStatus.append("\nWorking..." + values[0]);
}
```

```
    @Override
    protected void onProgressUpdate(Long... values) {
        super.onProgressUpdate(values);

        tvStatus.append("\nWorking..." + values[0]);
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);

        // We can use UI thread here
        if (pdWaiting.isShowing()) pdWaiting.dismiss();

        // Show done message
        tvStatus.append("\nEnd Time: " + System.currentTimeMillis());
        tvStatus.append("\nDone!");
    }
}
```

- 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);
            }
        }
```

4. Tự viết 1 ứng dụng sử dụng Asynctask để phát nhạc

5. Viết lại ứng dụng bài 4 sử dụng RxJava, Rx Android

IV. THAM KHẢO THÊM (Tự xem thêm)

1. 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**.

2. 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.

3. Calligraphy

- https://github.com/chrisjenx/Calligraphy

4. Drawable

- https://developer.android.com/guide/topics/resources/drawable-resource.html
- http://www.vogella.com/tutorials/AndroidDrawables/article.html