Fpt University

FPT POLYTECHNIC

www.poly.edu.vn

ANDROID NETWORK

BÀI 1: THEARD

Mục tiêu

- Tìm hiểu Thread - Ưu và khuyết điểm
- Sử dụng AsyncTask
- Class Handler

THREAD LÀ GÌ?

- Thread (luồng) là một đơn vị thực thi song song.
- Khi một chương trình được chạy sẽ có ít nhất 1 thread chính được chạy và thread có thể bật các thread bổ sung để phục vụ cho các mục đích cụ thể nào đó của thread chính.
- Từ đó ta có khái niệm Multi_threaded (đa luồng). Mỗi ứng dụng có thể thực hiện nhiều công việc đồng thời (mỗi công việc là một Thread)

LỢI ÍCH CỦA THREAD?

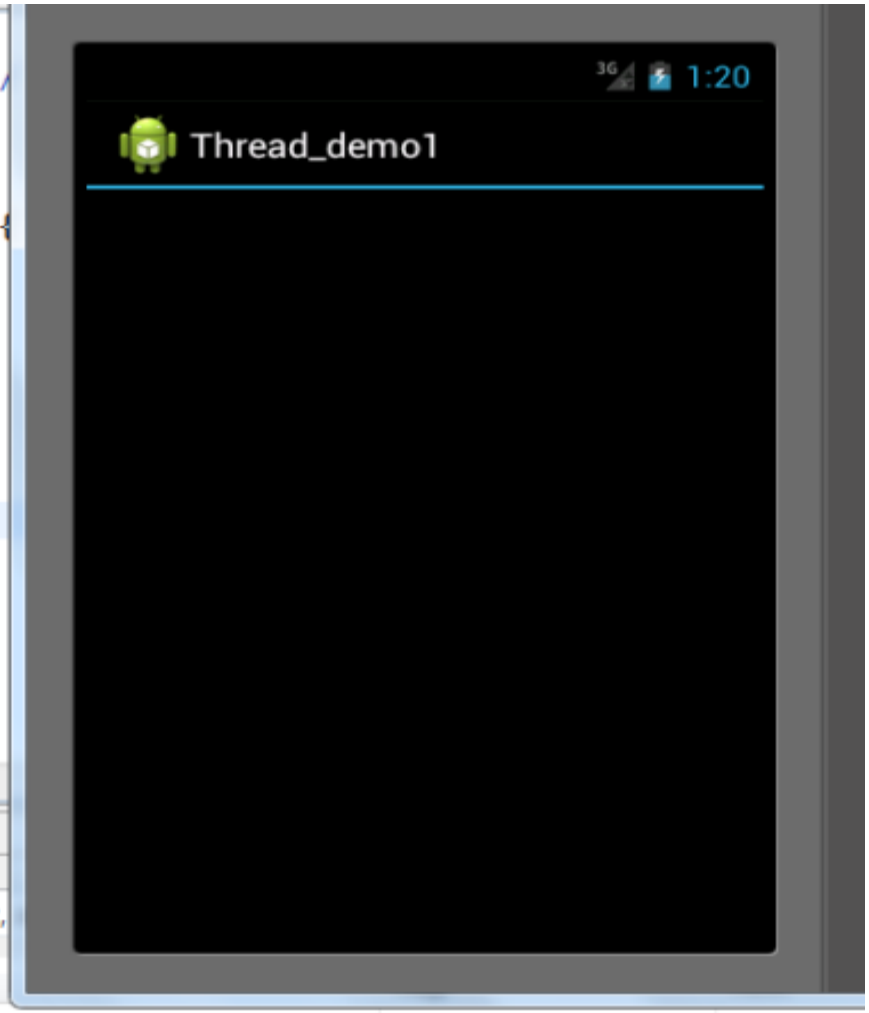
- Khi gặp các công việc độc lập thì đa luồng (Multi_threaded) là giải pháp hiệu quả giúp cho ứng dụng chạy nhanh hơn.

- Xét ví dụ sau: Ta có 02 công việc cần xử lý:
  1. Download hình ảnh
  2. Nhập dữ liệu khách hàng

  Do 02 công việc này độc lập nên ta có thể sử dụng đa luồng ở đây.

THREAD – VÍ DỤ

```
class Thread_demo1Activity extends Activity {
 Called when the activity is first created. */
tView tv1,tv2;

erride
lic void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  tv1=(TextView)findViewById(R.id.textView1);
  tv2=(TextView)findViewById(R.id.textView2);

  //cong viec 1 : tieu ton rat nhieu thoi gian
  for(int i=0;i<10000;i++)
      Log.d("vong lap", " "+i);

  //cong viec tiep theo:
  tv2.setText("Chao ban");
```



Search for messages. Accepts Java regexes. Prefix with pid:,

Level	Time	PID	Application		
D	10-10 13:20:3...	620	com.android.thread_demo1	vong lap	2136
D	10-10 13:20:3...	620	com.android.thread_demo1	vong lap	2137
D	10-10 13:20:3...	620	com.android.thread_demo1	vong lap	2138

THREAD – VÍ DỤ

- Theo ví dụ trên ta thấy:

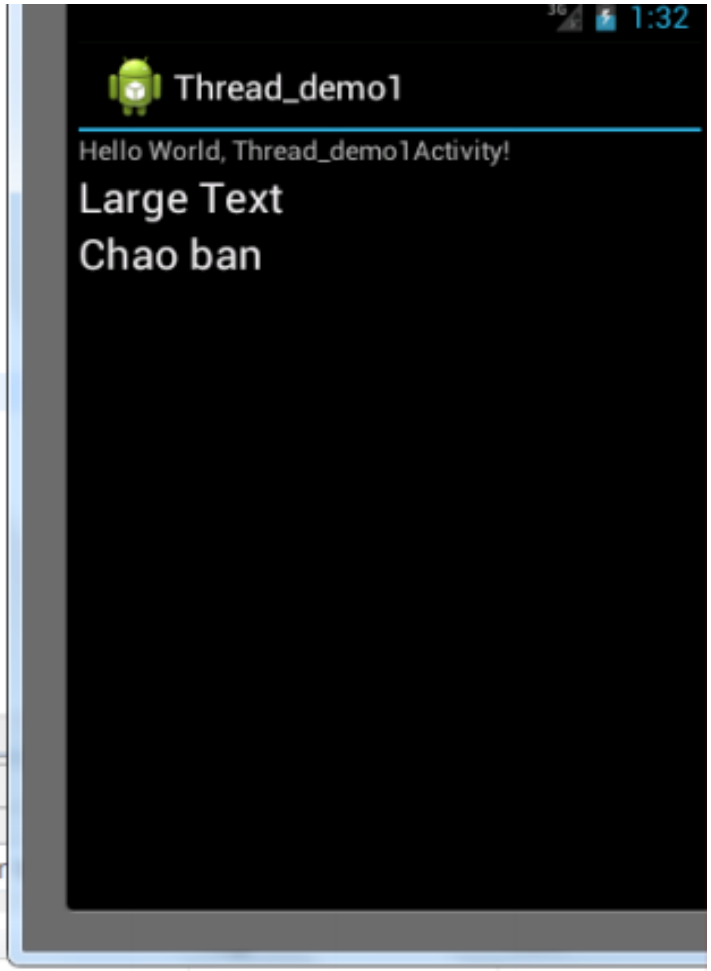
1 vòng for chạy 10 ngàn lần. Khi đó ta phải chờ cho khối lệnh đó thực thi xong mới chạy tiếp được.

Cụ thể tv2 không hiện được xuất ra text "Chao bạn" mà phải đợi for làm xong 10 ngàn lần.

Vậy ta có thể bỏ công việc chạy 10 ngàn làn vào 1 thread để chạy riêng thành 1 luồng khác. Khi đó tv2 sẽ được cập nhật liền mà không cần đợi.

THREAD – VÍ DỤ

```
Thread thread1=new Thread(new Runnable()
{

    public void run() {
        //cong viec 1 : tieu ton rat nhieu thoi gian
        for(int i=0;i<10000;i++)
            Log.d("luong chay"," " + i);

    }

});
thread1.start();

//cong viec tiep theo:
tv2.setText("Chao ban");

}
```



Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or

Level	Time	PID	Application		
D	10-10 13:32:0...	673	com.android.thread_demo1	luong chay	1904
D	10-10 13:32:0...	673	com.android.thread_demo1	luong chay	1905
D	10-10 13:32:0...	673	com.android.thread_demo1	luong chay	1906

SỬ DỤNG THREAD

- Cách 1: Kế thừa từ Class Thread

```
public class MyThread  extends Thread{

    @Override
    public void run() {
```

- Cách 2: Kế thừa từ lớp bất kì và thực thi giao diện Runnable

```
public class MyThread  extends View implements Runnable{
```

SỬ DỤNG THREAD
CÁCH 1: KẾ THỪA TỪ CLASS THREAD

Bước 1: Tạo 1 class mới MyThread.java như hình

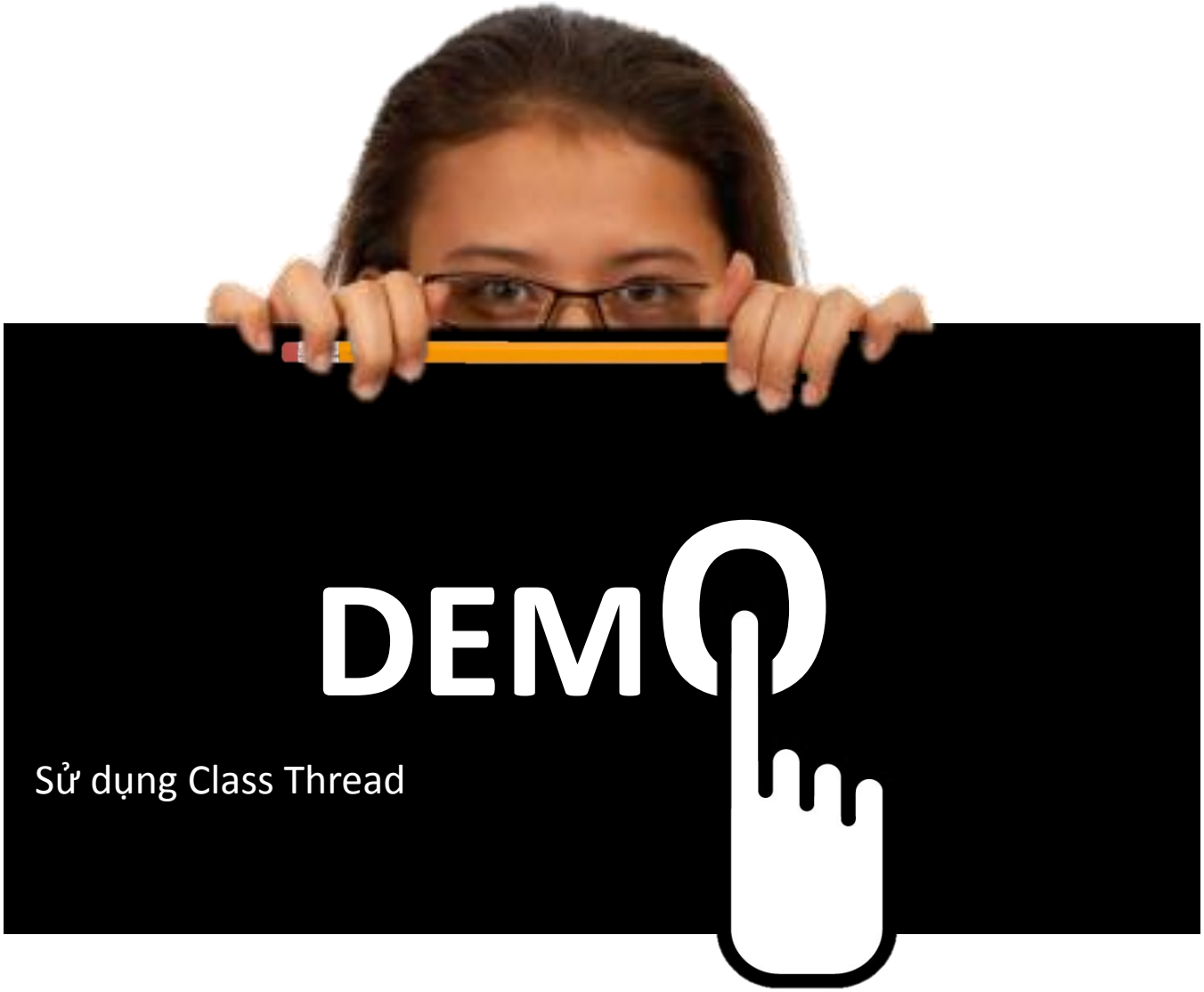
```
public class MyThread  extends Thread{

    @Override
    public void run() {
      //cong viec 1 : tieu ton rat nhieu thoi gian
            for(int i=0;i<10000;i++)
            {
                  Log.d("luong chay"," " + i);
            }

            super.run();
    }
}
```

Sử dụng Thread
Cách 1: Kế thừa từ Class Thread

Bước 2: Trong chương trình chính ta khai báo đối tượng a thuộc lớp MyThread (toàn cục).

Trong hàm onCreate tao khởi tạo đối tượng thread và dùng hàm start() để gọi thread chạy

```
a=new MyThread();
a.start();

//cong viec tiep theo:
tv2.setText("Chao ban");
```


DEMO
Sử dụng Class Thread

Sử dụng Thread
Cách 2: Interface Runnable

Bước 1: Ta tạo 1 class, có thể kế thừa từ 1 lớp bất kì nào đó sau đó thực thi giao diện "Runnable". Trong class ta phải thực thi hàm run cho giao diện.

```
public class MyThread  extends View implements Runnable{

    public MyThread(Context context) {
        super(context);
        // TODO Auto-generated constructor stub
    }

    public void run() {
        // TODO Auto-generated method stub
         //cong viec 1 : tieu ton rat nhieu thoi gian
        for(int i=0;i<10000;i++)
        {
            Log.d("luong chay"," " + i);
        }
    }
}
```

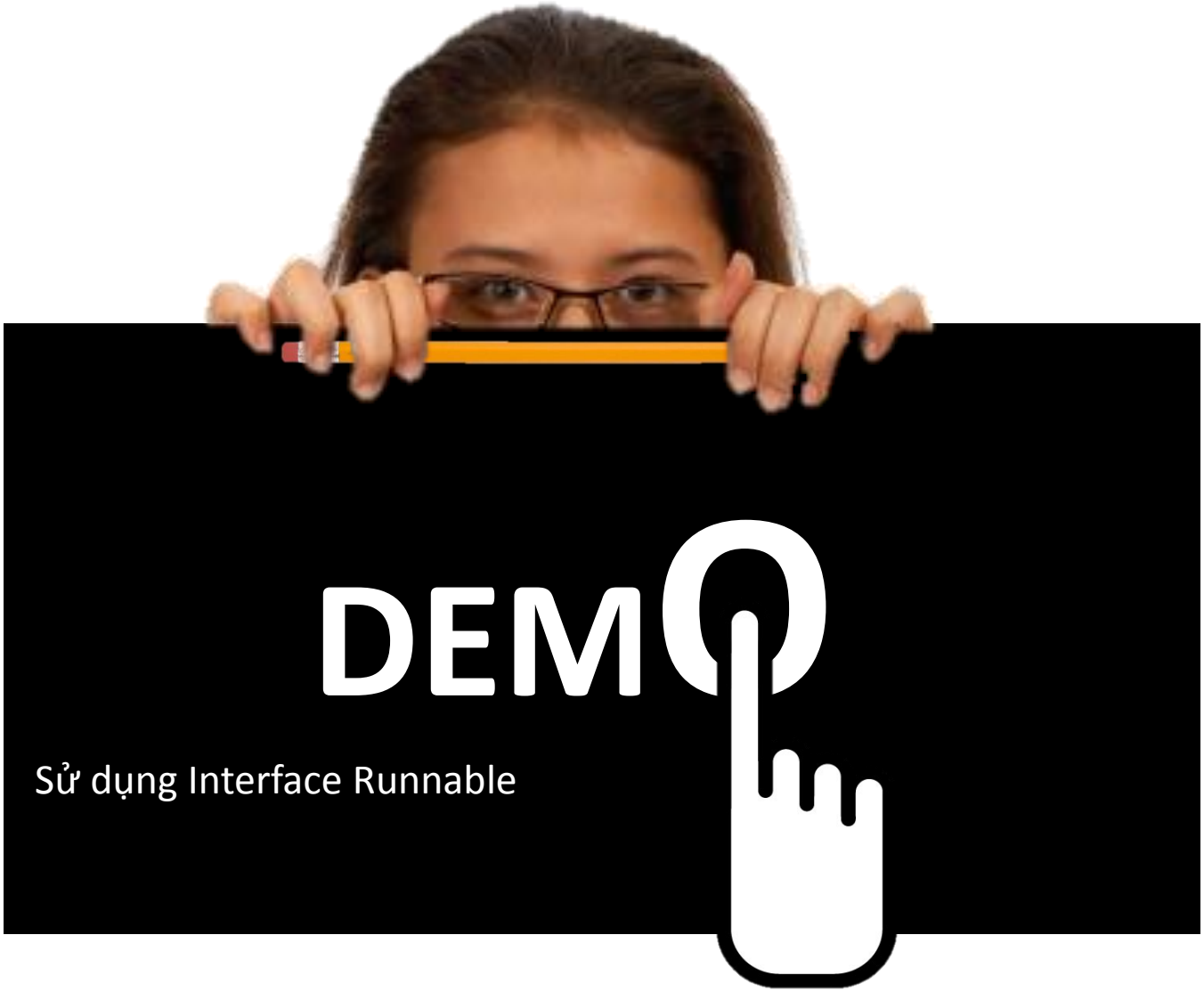
Chú ý: ta có thể kế thừa từ nhiều class ở đây ta kế thừa từ View, class View yêu cầu phải có hàm tạo nên ta viết hàm tạo. Kế thừa từ class nào là tùy thuộc theo bài toán, hoặc không kế thừa cũng được. Chỉ cần để ý implement Runnable và hàm run.

SỬ DỤNG THREAD
CÁCH 2: INTERFACE RUNNABLE

Bước 2: Ta tạo đối tượng thuộc class ta vừa xây, sau đó tạo ra 1 đối tượng Thread và truyền vào đối tượng của ta, cuối cùng thì dùng start() để chạy.

```
a=new MyThread(this);//tao ra doi tuong Runnable
Thread t=new Thread(a);//tao doi tuong thread
t.start();

//cong viec tiep theo:
tv2.setText("Chao ban");
```

- Không gọi run() mà gọi start() vì start() sẽ tạo ra luồng mới, cấp phát bộ nhớ và gọi hàm run() để chạy, ta không nên gọi trực tiếp hàm run.
- Tại sao ta phải cách thực thi giao diện Runnable? Vì java không cho đa thừa kế do đó ta muốn kế thừa từ 1 class nào đó và xem nó như luồng thì ta phải dùng cách 2.


DEMO
Sử dụng Interface Runnable

VẤN ĐỀ KẾT THÚC THREAD

- Sử dụng biến Boolean để kiểm tra main Thread còn chạy?

```
public class MyThread  extends View implements Runnable{

    boolean running=true; //bien de dung luong
    public MyThread(Context context) {
            super(context);
    }

    public void run() {
            // TODO Auto-generated method stub
             //cong viec 1 : tieu ton rat nhieu thoi gian
            for(int i=0;i<10000;i++)
            {
                    if(running==false) //neu bien running=flase se vang khoi for, ra khoi run va
ket thuc
                            break;
                    Log.d("luong chay"," " + i);
            }
    }
}
```

- Quay về class chính override lên hàm onDestroy

```
@Override
protected void onDestroy() {
        // TODO Auto-generated method stub
        super.onDestroy();
        if(t.isAlive())
                a.running=false;
}
```

VẤN ĐỀ UPDATE GIAO DIỆN

- Android có 2 luật quan trọng cho thread:
  - Không khóa UI thread.
  - Không truy cập Android UI toolkit bên ngoài UI thread.

- Không thể truy cập đến giao diện từ ngoài luồng giao diện (luồng giao diện ở đây là file java chính). Nói đơn giản không cập nhật View ngoài UI Thread(main Thread).

- Để thực hiện việc cập nhật UI từ những thread khác Android cung cấp các cách sửa như:
  - Activity.runOnUiThread(Runnable)
  - View.post(Runnable)
  - View.postDelayed(Runnable, long)

VẤN ĐỀ UPDATE GIAO DIỆN

```
public void run() {
        // TODO Auto-generated method stub
        for(int i=0;i<10000;i++)
        {
                if(running==false) //neu bien running=flase se vang khoi for,
break;
                Log.d("luong chay"," " + i);
        }
        tv.post(new Runnable(){

                public void run() {
                        // TODO Auto-generated method stub
                        tv.setText("da ket thuc luong");
                }
        });
}
```


DEMO
Cập nhật giao diện

VẤN ĐỀ UPDATE GIAO DIỆN

Tuy nhiên cách này xem ra khá phức tạp khó viết và khó bảo trì.

Để điều khiển nhiều thao tác phức tạp với thread con ta nên dùng lớp Handler trong lớp con để xử lý các thông điệp từ UI thread.

Cách tốt nhất là kế thừa lớp AsyncTask để luồng con có thể tương tác với UI.

SỬ DỤNG ASYNCTASK

- Class này cho phép thực hiện công việc bất đồng bộ lên giao diện người dùng. Nó thực hiện thao tác khóa trong luồng con và sau đó cập nhật kết quả lên **UI thread**.
- Để sử dụng cần tạo ra **subclass** (không tách nó thành 1 file java khác) kế thừa từ **AsyncTask** sau đó thực thi hàm **doInBackground()**. Để cập nhật giao diện ta thực thi hàm **onPostExcute()**, hàm này sẽ chuyển giao kết quả từ **doInBackground()** và chạy trên UI thread vì thế ta có cách an toàn để cập nhật UI.
- Phương thức **execute()** gọi doInBackground() và onPostExecute().

SỬ DỤNG ASYNCTASK

- Phương thức **doInBackground()** chứa mã sẽ thực thi trong background thread. Phương thức này chạy tự động trong một luồng riêng.
- Phương thức **onPostExcute()** đồng bộ nó với UI thread và cho phép update UI. Phương thức này sẽ chạy khi phương thức doInBackground() kết thúc.
- Class này dùng generic như sau:
  AsyncTask<Params,ProgessValue,ResultValue>
  trong đó:
  - Params : truyền vào hàm doInBackground() như input.
  - ProgessValue: dùng để xử lý thông tin.
  - ResultValue: trả về từ doInBackground() và truyền vào onPostExecute() như đối số.

SỬ DỤNG ASYNCTASK

```
class ChayNen extends AsyncTask<Integer,Integer,String>
  {
        @Override
        protected String doInBackground(Integer... params) {
            // TODO Auto-generated method stub
            for(int i=0;i<params[0];i++)
            {
                Log.d("luong", " " + i);
                publishProgress(i);
            }
            return "da xong";
        }

        @Override
        protected void onPostExecute(String result) {
            // TODO Auto-generated method stub
            super.onPostExecute(result);
            tv1.setText("luong ngam " + result);
        }

        @Override
        protected void onProgressUpdate(Integer... values) {
            // TODO Auto-generated method stub
            super.onProgressUpdate(values);
            tv1.setText("chay" + values[0]);
        }

  }
```

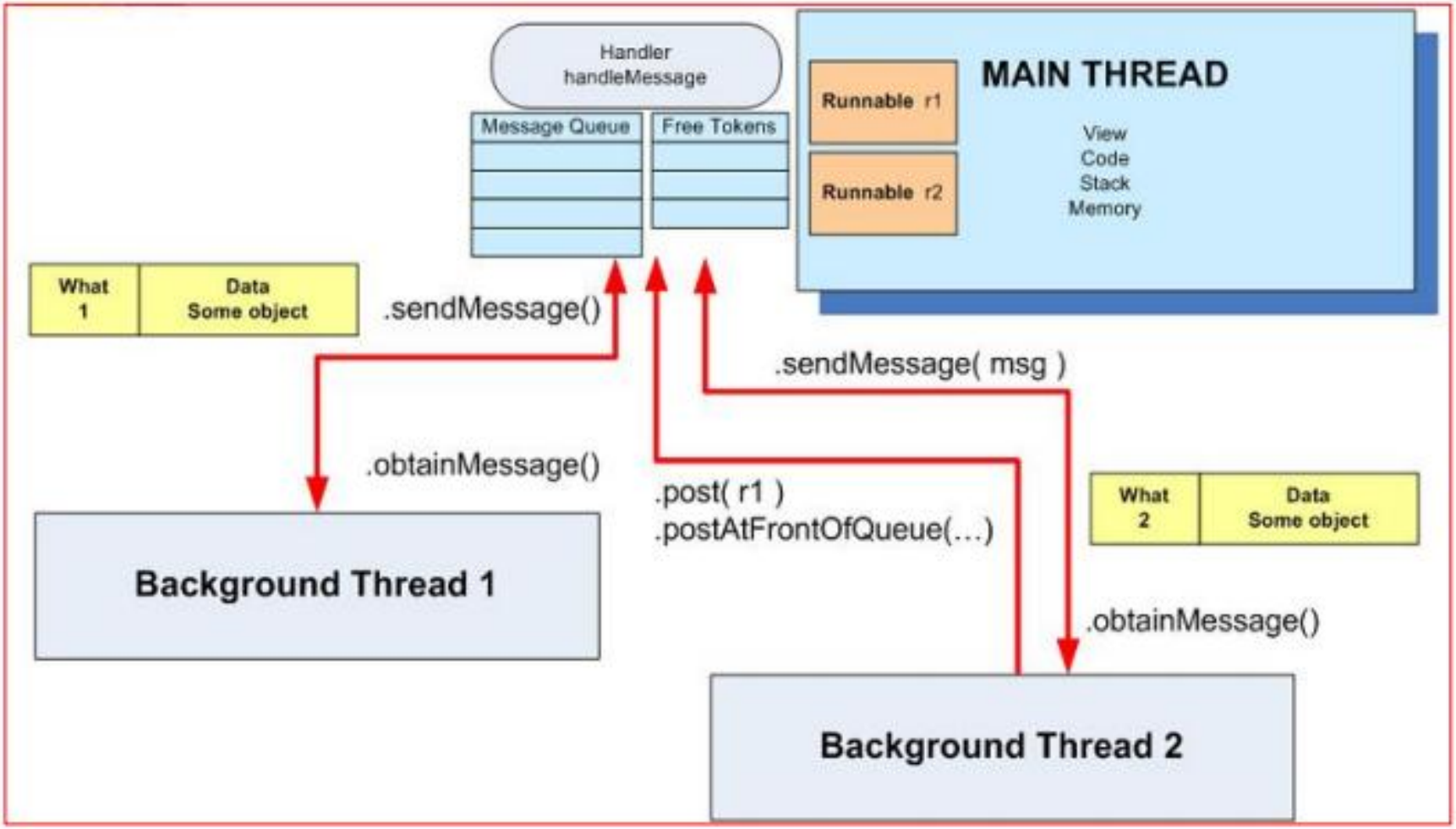

DEMO
Sử dụng AsyncTask

SỬ DỤNG CLASS HANDLER

- Ta có thể tạo ra các thread con tương tác với thread chính thông qua một Handler.

- Khi handler được tạo, nó gắn với MessageQueue (hàng đợi thông điệp) của thread tạo ra nó. Từ đó nó sẽ gởi các các thông điệp tới hàng đợi thông điệp và thực thi chúng khi chúng ra khỏi MessageQueue

SỬ DỤNG CLASS HANDLER



SỬ DỤNG CLASS HANDLER

- Các thread con cần liên lạc với Thread chính sẽ dùng phương thức obtainMessage() để lấy 1 message token (giống như mua vé).
- Sau khi có message token thread con sẽ ghi dữ liệu vào message token đó và gởi nó về Handler để đưa nó vào cuối MessageQueue bằng phương thức sendMessage().
- MessageQueue làm việc theo cơ chế FIFO.
- Handler dùng phương thức handleMessage() để xử lý các message mới được gởi tới thread chính.
- Các Message lấy ra từ MessageQueue có thể trả dữ liệu cho thread chính hoặc yêu cầu thực thi các đối tượng Runnable qua phương thức post().

Sử dụng Class Handler Phương pháp post

Thread chinh

```
public class Thread_handler1Activity extends Activity {
    /** Called when the activity is first created. */
     TextView tv1,tv2;
     Handler handler=new Handler();

    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       tv1=(TextView)findViewById(R.id.textView1);
       tv2=(TextView)findViewById(R.id.textView2);

       Thread a=new Thread(backgroundtask);
       a.start();
    }
```

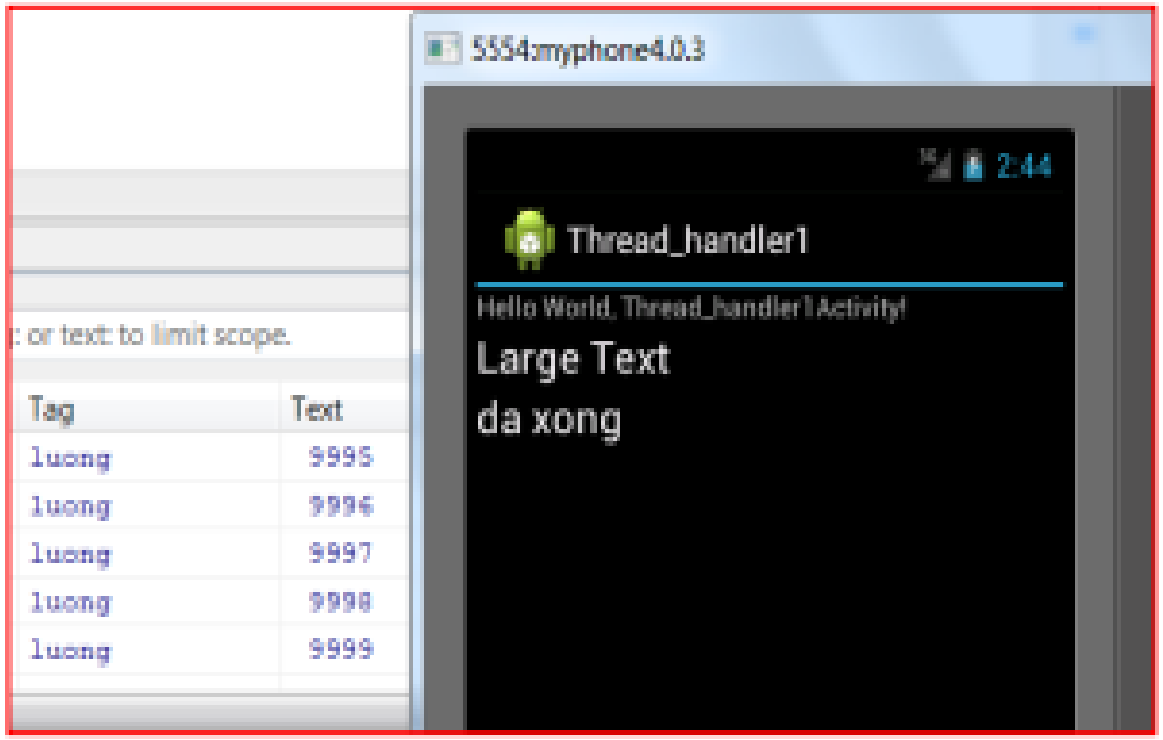
Background thread

```
    Runnable backgroundtask=new Runnable() {

            public void run() {
                    // TODO Auto-generated method stub
                    for(int i=0;i<10000;i++)
                    {
                            Log.d("luong"," "+ i);
                    }
                    handler.post(updateUItask);
            }
     };
```

Sử dụng Class Handler Phương pháp post

Runnable hỗ trợ cập nhật UI

```
Runnable updateUItask=new Runnable() {

        public void run() {
                // TODO Auto-generated method stub
                tv2.setText("da xong");
                //Log.d("luong","ket thuc");
        }
};
```



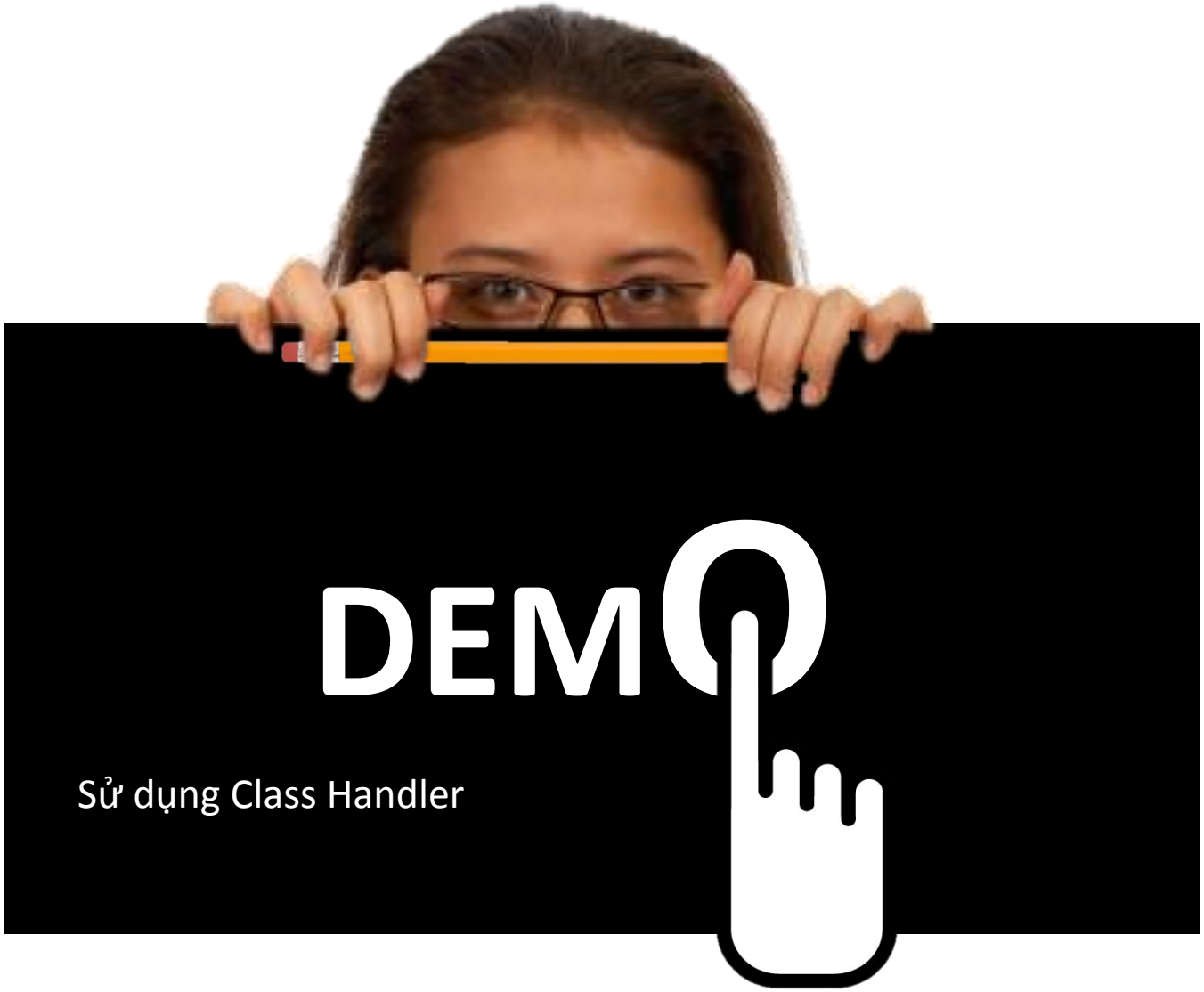
Sử dụng Class Handler
Phương pháp dùng Message

Main thread

```
//.....
    Handler handler;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      tv1=(TextView)findViewById(R.id.textView1);
      tv2=(TextView)findViewById(R.id.textView2);

      handler=new Handler(){
            @Override
            public void handleMessage(Message msg) {
                  // TODO Auto-generated method stub
                  super.handleMessage(msg);
                  tv2.setText("xong");
            }
      };

      mythread.start();
   }
```

Background thread

```
Thread mythread=new Thread(new Runnable(){
            public void run() {
                  // TODO Auto-generated method stub
                  for(int i=0;i<10000;i++)
                        Log.d("luong", " " + i);
                  Message me=handler.obtainMessage();
                  handler.sendMessage(me);
            }

   });
```


DEMO
Sử dụng Class Handler

TỔNG KẾT NỘI DUNG BÀI HỌC

- Tìm hiểu Thread - Ưu và khuyết điểm
- Các vấn đề
- Sử dụng AsyncTask
- Class Handler