Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
**TRUNG TÂM TIN HỌC**

# Lập trình Android

## Bài 28: *Tích hợp Java 8, RxJava*

**Phòng LT & Mạng**

*http://csc.edu.vn/lap-trinh-va-csdl*

**2019**

# Nội dung

1. Java 8 Lamda Expression
2. RxJava

# Java 8 - Lambda Expression

- **Java 8 Lambda expression** giúp source code đơn giản, ngắn gọn và dễ đọc hơn. Đặc biệt trong các trường hợp event listener chỉ có một phương thức cần implement

- Ví dụ, đoạn code sau xử lý sự kiện onClick lên một button:

```
btnClick.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Log.i("CSC", "Button was clicked");
    }
});
```

- Khi sử dụng Lambda expression, đoạn code thay thế như sau:

```
btnClick.setOnClickListener(v -> Log.i("CSC", "Button was clicked"));
```

- Trường hợp có nhiều hơn 1 dòng code xử lý, sử dụng khối { } để bao bọc:

```
btnClick.setOnClickListener(v -> {
    Log.i("CSC", "Button was clicked");
    Toast.makeText(MainActivity.this, "Clicked", Toast.LENGTH_SHORT).show();
});
```

# Java 8

- Để sử dụng cú pháp của Java 8 trong project Android, cần cấu hình **build.gradle**

```
android {
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
```

# Java 8 - Method references

- Trong một số trường hợp Lambda expression có thể chứa dấu bốn chấm **::** để tham chiếu đến phương thức xử lý một sự kiện:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    // …
    btnClick.setOnClickListener(this::handleButtonClick);
}

private void handleButtonClick(View v) {
    Log.i("CSC", "Button was clicked");
    Toast.makeText(MainActivity.this, "Clicked", Toast.LENGTH_SHORT).show();
}
```

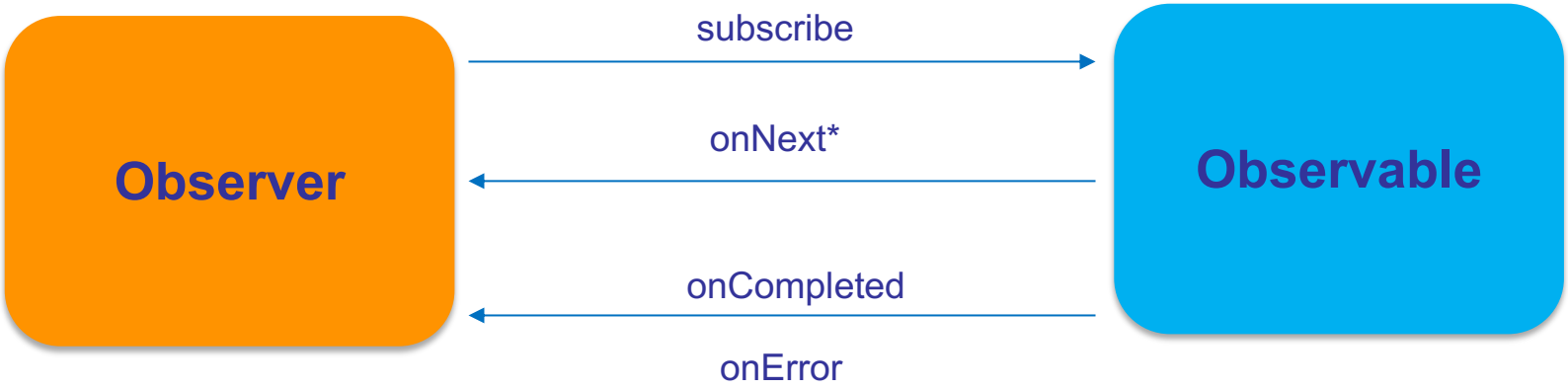
# RxJava

- **RxJava** trên Android là thư viện giúp thực hiện chuỗi các thao tác bất đồng bộ
- Xử lý các thao tác đồng thời tốt hơn so với **AsyncTask**
- Phát hiện lỗi sớm hơn so với **AsyncTask**
- Giúp viết code rõ ràng, bớt rắc rối và dễ hiểu hơn
- Trong bài học này, **RxJava** được minh hoạ tích hợp với **Retrofit** giúp xâu chuỗi các lời gọi Web Services API
- Tích hợp vào project:
  - Khai báo **dependencies** trong **build.gradle**

```
dependencies {
    implementation 'io.reactivex:rxandroid:1.2.0'
    implementation 'io.reactivex:rxjava:1.1.4'
}
```

# Observable và Observer

- **Observable** phát ra các đối tượng
- **Observer** "tiêu thụ" (consume) các đối tượng do **Observable** phát ra
- **Observable** chưa thể phát ra các đối tượng khi chưa có đăng ký (**subscribe**) với một **Observer** nào




- Khi **Observable** phát ra 1 đối tượng, hàm **onNext()** của **Observer** sẽ được gọi. Hàm này được gọi 0 hoặc nhiều lần tuỳ vào số lần phát ra đối tượng của **Observable**
- Khi **Observer** tiêu thụ xong các đối tượng, hàm **onCompleted**() sẽ được gọi. Nếu có lỗi xảy ra trong quá trình này, hàm **onError**() sẽ được gọi. Sau 2 hàm này, sẽ không có đối tượng nào được phát ra từ **Observable**

# Định nghĩa một Observable

- Ví dụ sau tạo ra một **Observable**, phát ra các đối tượng **String**:

```
Observable<String> myObservable = Observable.create(
        new Observable.OnSubscribe<String>() {
            @Override
            public void call(Subscriber<? super String> sub) {
                // Phát ra các đối tượng dữ liệu tới Subscriber (Observer)
                sub.onNext("a");
                sub.onNext("b");
                sub.onNext("c");
                // Thông báo kết thúc sự kiện
                sub.onCompleted();
            }
        }
);
```

# Định nghĩa một Observer

- Tiếp theo, định nghĩa một **Observer** để "tiêu thụ" các đối tượng dữ liệu cho **Observable** phát ra:

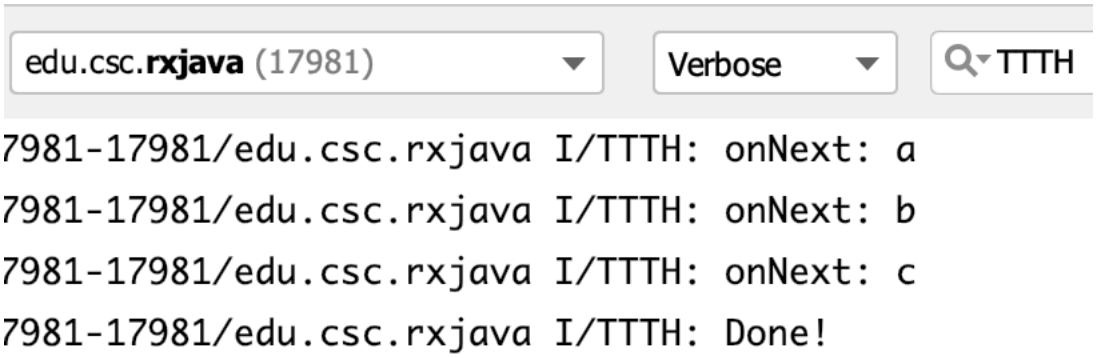
```
Observer<String> mySubscriber = new Observer<String>() {
    // Được gọi khi có mỗi đối tượng dữ liệu được phát ra
    @Override
    public void onNext(String s) {
        Log.i("TTTH", "onNext: " + s);
    }
    // Được gọi khi một Observable hoàn tất
    @Override
    public void onCompleted() {
        Log.i("TTTH", "Done!");
    }
    // Được gọi khi có lỗi xảy ra trong quá trình thực hiện
    @Override
    public void onError(Throwable e) { }
};
```

# Đăng ký Observable với Observer

- Thực hiện **đăng ký** Observable với Observer:

  myObservable.**subscribe**(mySubscriber);

- Kết quả chạy thử:




- Trong ví dụ này, các đối tượng String được Observable phát ra và được Observer in ra Logcat trong hàm **onNext** của Observer. Khi tất cả các đối tượng String đã được xử lý, hàm **onCompleted** của Observer sẽ được gọi.

# Các khác để tạo đối tượng Observable

- Có thể tạo đối tượng **Observable** từ các đối tượng dữ liệu rời rạc:

```
Observable.just("a", "b", "c")
```

- Và sau đó đăng ký với một đối tượng **Observer**:

```
Observable.just("a", "b", "c").subscribe(new Observer<String>() {
    @Override
    public void onNext(String s) { Log.i("TTTH", "onNext: " + s); }

    @Override
    public void onCompleted() { Log.i("TTTH", ”D one!"); }

    @Override
    public void onError(Throwable e) { }
});
```

# Các khác để tạo đối tượng Observable (2)

- Cũng có thể tạo ra đối tượng **Observable** từ một danh sách:

```
ArrayList<String> items = new ArrayList<>();
items.add("red");
items.add("orange");
items.add("yellow");

Observable.from(items);
```

# Schedulers

- Mặc định **RxJava** hoạt động theo cơ chế đồng bộ, nhưng có thể hoạt động bất đồng bộ sử dụng **Scheduler**. Ví dụ, có thể định nghĩa các network request hoạt động ở background thread, nhưng các hàm callback tương ứng (khi network request kết thúc, hoặc khi có lỗi) cần được thực hiện ở main thread (UI thread).

```
Observable.from(Arrays.asList("a", "b", "c", "d", "e"))
    .subscribeOn(Schedulers.newThread()) // các công việc của Observable được thực hiện ở thread mới
    .observeOn(AndroidSchedulers.mainThread()) // kết quả thực thi trả về cho main UI thread (onNext)
    .subscribe(new Observer<String>() { // đăng ký Observable với Observer
  @Override
  public void onNext(String s) { Log.i("TTTH", "onNext: " + s); }

  @Override
  public void onCompleted() { Log.i("TTTH", "done!"); }

  @Override
  public void onError(Throwable e) { }
});
```

# Các loại Scheduler

- **Scheduler** chịu trách nhiệm thực hiện các hoạt động của **Observable** trên các **thread** khác nhau dựa trên phương thức **subcribeOn** và **observeOn**. Một số loại **Scheduler** phổ biến:

| **Loại Scheduler** | **Mô tả** |
|---|---|
| Schedulers.computation() | Xử lý các công việc tính toán, có thread-pool giới hạn theo số lượng CPU |
| Schedulers.immediate() | Sử dụng thread hiện tại |
| Schedulers.io() | Tạo thread mới, được hỗ trợ bởi thread-pool, sử dụng trong các tác vụ tốn thời gian như network request, read/write file |
| Schedulers.newThread() | Tạo ra một thread mới |
| Schedulers.trampoline() | Sắp xếp các công việc vào hàng đợi của thread hiện tại, thực hiện từng công việc một |

# Sử dụng RxJava với Retrofit

- **RxJava** có thể sử dụng với **Retrofit** giúp tạo ra chuỗi các API request
- Thư viện **Retrofit** bao bọc API request bởi một **Observable** để có thể sử dụng với **RxJava**
- Cấu hình **build.gradle** dependencies:

```
dependencies {
    implementation 'com.squareup.retrofit2:retrofit:2.6.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.6.0'
    implementation 'io.reactivex:rxandroid:1.2.0'
    implementation 'io.reactivex:rxjava:1.1.4'
    implementation 'com.squareup.retrofit2:adapter-rxjava:2.0.2'
}
```

# Sử dụng RxJava với Retrofit (2)

- Định nghĩa API endpoint, ví dụ: API lấy thông tin một sản phẩm trong minh hoạ bài học Web Service:

(1) Thay thế **Call** bởi **Observable**

(2) Đăng ký sử dụng **RxJava** với **Retrofit**

```
public interface ProductService {
    @GET("product/{id}")
1   Observable<Product> getProduct(@Path("id") int id);
}
```

- Kết nối đối tượng **Retrofit** với **RxJava** bằng phương thức **addCallAdapterFactory**:

```
public class MainActivity extends AppCompatActivity {
    ProductService service;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // …
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl("http://10.0.2.2:5000/")
                .addConverterFactory(GsonConverterFactory.create())
2               .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
                .build();
        service = retrofit.create(ProductService.class);
    }
```

# Sử dụng RxJava với Retrofit (3)

- Thực hiện lấy thông tin 1 sản phẩm biết mã sản phẩm (id):

```
private void getProduct(int id) {
    Observable<Product> call = service.getProduct(id);

    call.subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(new Subscriber<Product>() {
                @Override
                public void onCompleted() {

                }

                @Override
                public void onError(Throwable e) {

                }

                @Override
                public void onNext(Product product) {
                  // product - kết quả sau khi thực hiện API request
                }
            });
}
```

# Sử dụng RxJava với Retrofit (4)

- Thực hiện lấy thông tin nhiều sản phẩm biết danh sách các id của các sản phẩm. Ví dụ, một đối tượng người dùng có một danh sách các sản phẩm yêu thích, ứng dụng cần lấy về danh sách này từ server:

```
private void getProductArray(ArrayList<Integer> id) {
    List<Observable<Product>> calls = new ArrayList<>(); // tạo danh sách các Observable
    for (int i : id) {
        Observable<Product> call = service.getProductRx(i);
        calls.add(call);
    }

    Observable.merge(calls).subscribeOn(Schedulers.io()) // xâu chuỗi các Observable với hàm merge
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(new Subscriber<Product>() {
                @Override
                public void onCompleted() {
                }

                @Override
                public void onError(Throwable e) {
                }

                @Override
                public void onNext(Product product) {
                    // xử lý với mỗi product kết quả từ API, ví dụ: thêm vào danh sách các products
                }
            });
}
```

Trong trường hợp này, **RxJava** giúp thực hiện riêng lẻ API lấy thông tin 1 sản phẩm trên danh sách các id của các sản phẩm. Kết quả mỗi lần gọi API trả về trong hàm **onNext**(…). Khi thực hiện xong chuỗi công việc này, hàm **onCompleted**() sẽ được gọi.

# Q&A