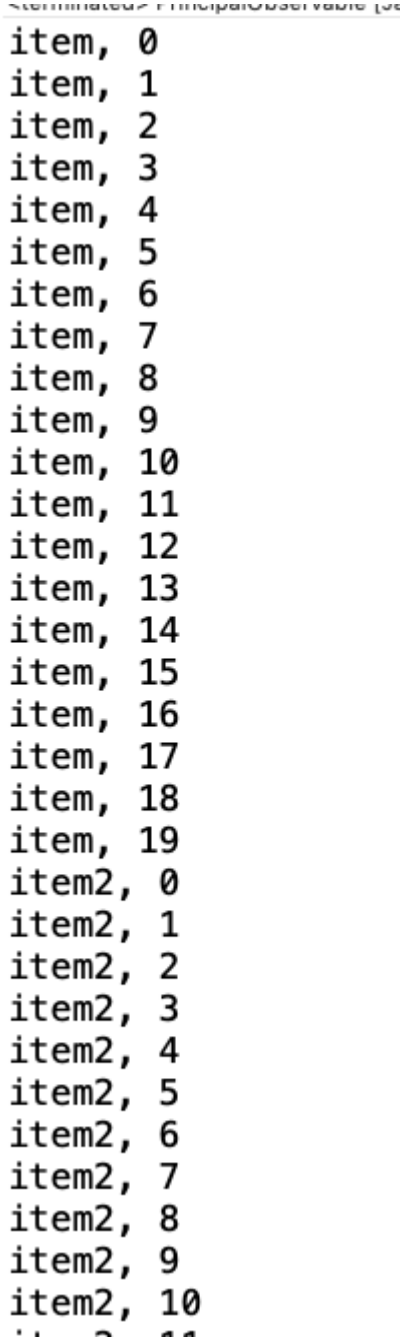
El concepto de ConnectableObservable de RxJava es uno de los conceptos que a veces más cuesta entender a nivel de RxJava y sus observables . Un ConnectableObservable es un Observable Hot o observable caliente. Por defecto todos los observables son frios. ¿Qué quiere decir esto? . Vamos a ver un ejemplo sencillo de programación asíncrona en el que veamos como se comporta un observable frio.

```
package com.arquitecturajava.ejemplo001;

import java.util.concurrent.TimeUnit;

import io.reactivex.Observable;
import io.reactivex.observables.ConnectableObservable;

public class PrincipalObservable {

  public static void main(String[] args) {

    Observable<Long> frios = Observable.interval(500,
TimeUnit.MILLISECONDS).take(20);
    frios.subscribe(l -> System.out.println("item, " + l));
    try {
      Thread.sleep(10000);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    frios.subscribe(l -> System.out.println("item2, " + l));
    try {
      Thread.sleep(10000);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
```

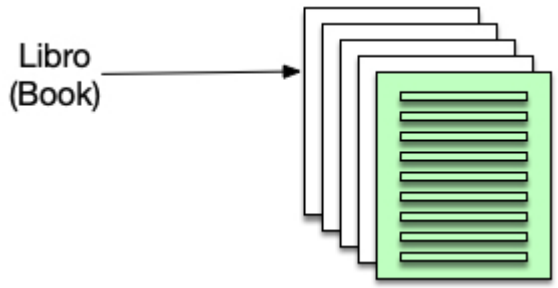
```
      e.printStackTrace();
    }
  }
}
```

En este caso estamos utilizando un observable frio clásico es decir emite los elementos de uno en uno y cada vez que nos subscribamos obtendremos todos los elementos que pertenecen al observable desde el principio. Si ejecutamos el código podremos obtener algo como lo siguiente:



Cuando nos conectamos a un observable frio y nos subscribimos automáticamente recibimos toda la secuencia de elementos desde el principio. Esto nos puede parecer extraño pero es muy sencillo de entender . El simil le podemos hacer con el concepto de un libro . Cuando a nosotros alguien nos regala un libro siempre podemos volverle a leer y por lo tanto siempre

nos dará toda la información desde el principio . Aunque tardamos un tiempo en acabarlo siempre le tenemos disponible para leerle desde cero.



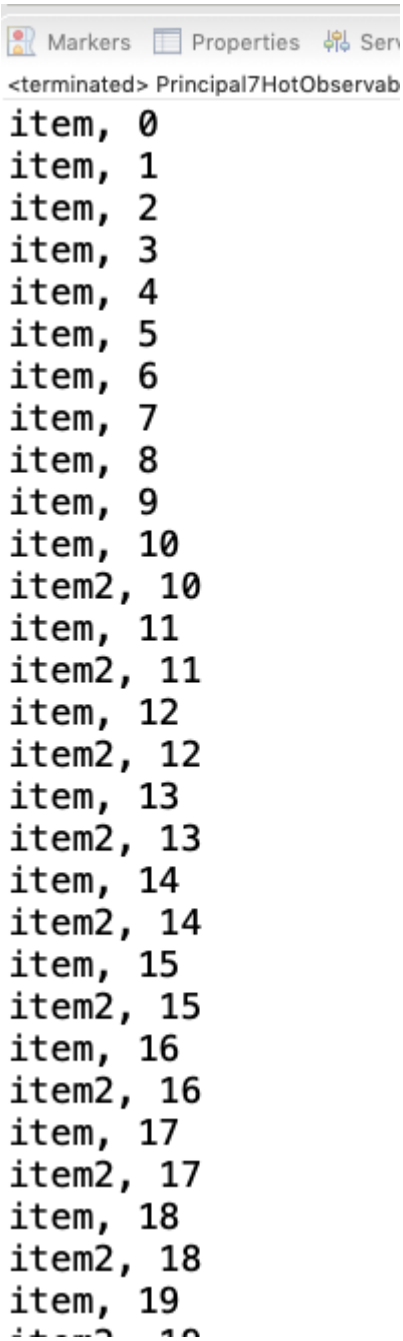
ConnectableObservable (Hot)

En este caso estamos ante una situación diferente cuando nosotros con RxJava activamos un ConnectableObservable estamos ante un Hot Observable en este caso no nos emitirá todos los elementos desde el principio cuando nos volvamos a subscribir . Sino que al subscribirnos por segunda vez solo recibiremos los items que quedan por emitir. Esta situación es idéntica a cuando alguien nos regala una entrada al cine para ver la película . Si llegamos tarde habrá una parte de la película que ya no podremos ver y que nos hemos perdido. Este es el concepto de ConnectableObservable y su funcionamiento . Vamos a verlo en código para que nos quede más claro.

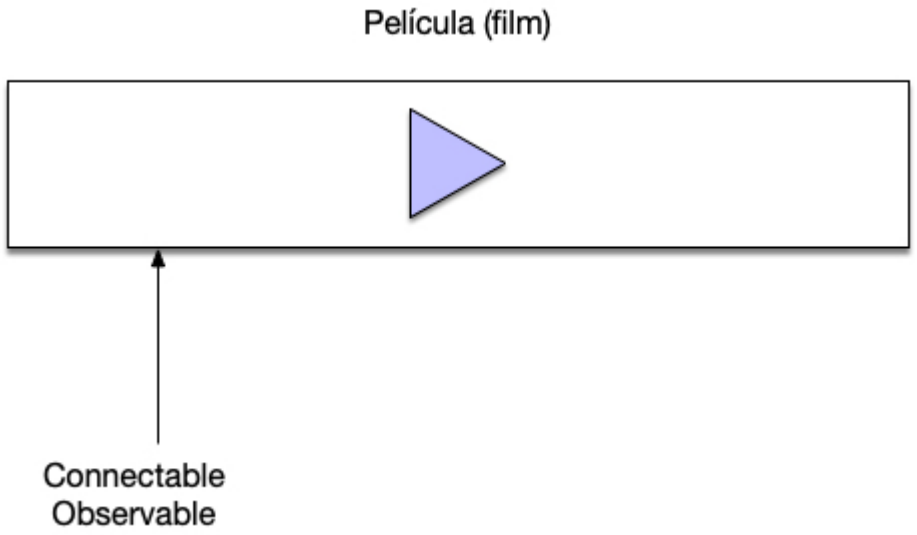
```
package com.arquitecturajava.ejemplo001;

import java.util.concurrent.TimeUnit;

import io.reactivex.Observable;
import io.reactivex.observables.ConnectableObservable;

public class Principal7HotObservable {

  public static void main(String[] args) {
```

```
    ConnectableObservable<Long> caliente = Observable.interval(500,
TimeUnit.MILLISECONDS).take(20).publish();
    caliente.connect();

    caliente.subscribe(l -> System.out.println("item, " + l));
    try {
      Thread.sleep(5000);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    caliente.subscribe(l -> System.out.println("item2, " + l));

    try {
      Thread.sleep(10000);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
}
```

En este caso si ejecutamos el código nos podremos dar cuenta que la segunda subcripción no recibe todos los elementos sino aquellos que todavía faltaban por emitir.



En este caso estamos en la situación de ver una película y habernos perdido una parte de ella ya que la segunda subscripción comienza en el item 10:



Ambos concepto de Observables son importantes para trabajar en el día a día.

Otros artículos relacionados

1. Stream vs Observable y sus diferencias
2. Promise vs Observable en JavaScript
3. Usando Rx Observables en JavaScript
4. Flux vs Mono ,Spring y la programación reactiva
5. RxJava