//onComplete - finishing all, no methods will be alive any more  
  
 // default subject can add elements and set listener  
 /\*PublishSubject<Integer> subject = PublishSubject.create();  
 subject.onNext(1);  
 subject.subscribe(System.out::println);  
 subject.onNext(2);  
 subject.onNext(3);  
 subject.onNext(4);\*/  
  
 // cashing all parameters  
 // and after adding new subject it adds to new one all cashed parameters  
 /\*ReplaySubject<Integer> s = ReplaySubject.create(); // default one  
 ReplaySubject<Integer> s = ReplaySubject.createWithSize(3); // with limited size  
 ReplaySubject<Integer> s = ReplaySubject.createWithTime(150, TimeUnit.MILLISECONDS, Schedulers.immediate()); //with limited time  
 ReplaySubject<Integer> s = ReplaySubject.createWithTimeAndSize(150, TimeUnit.MILLISECONDS, Schedulers.immediate(), 4); //limited both  
  
 s.subscribe(v -> System.out.println("Early:" + v));  
 s.onNext(0);  
 s.onNext(1);  
 s.subscribe(v -> System.out.println("Late: " + v));  
 s.onNext(2);\*/  
  
 // cashing only LAST parameter and works as ReplayingSubject with limit of 1 parameter  
 // default value can set first in order to get default action to all subscribers  
 /\*BehaviorSubject<Integer> s = BehaviorSubject.create();  
 s.onNext(0);  
 s.onNext(1);  
 s.onNext(2);  
 s.subscribe(v -> System.out.println("Late: " + v));  
 s.onNext(3);\*/  
  
 //stores only last value and activate action only before destroying  
 //if you don't use onComplete it won't do anything  
 /\*AsyncSubject<Integer> s = AsyncSubject.create();  
 s.subscribe(v -> System.out.println(v));   
 s.onNext(0);  
 s.onNext(1);  
 s.onNext(2);  
 s.onComplete();\*/  
  
  
  
  
 /\*Subject<Integer> s = ReplaySubject.create(); // Subject is a base class  
 s.subscribe(  
 v -> System.out.println(v),  
 e -> System.err.println(e) // for error, if delete this field you can't use onError  
 () -> System.out.println("Done")); // for onComplete method  
 s.onNext(0);  
 s.onError(new Exception("Oops"));\*/  
  
 // get variable, which connecting action and subscription  
 // to be able to unsubscribe or check if subscribed  
/\*Subject<Integer> values = ReplaySubject.create();  
 Disposable disposable = values.subscribe(  
 v -> System.out.println(v),  
 e -> System.err.println(e),  
 () -> System.out.println("Done")  
 );  
 values.onNext(0);  
 values.onNext(1);  
 disposable.dispose(); // unsubscribe  
 values.onNext(2);\*/

Using action with already defined values

or with time and period of recalling

//just creates observable, which will use defined values and onComplete after subscribing  
/\*Observable<String> values = Observable.just("one", "two", "three");  
Disposable disposable = values.subscribe(  
 v -> System.out.println("Received: " + v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
);\*/  
  
//empty creates observable, which use onCompleted after subscribing and dying  
/\*Observable<String> values = Observable.empty();  
Disposable disposable = values.subscribe(  
 v -> System.out.println("Received: " + v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
);\*/  
  
//error returns observable without doing anything after subscribing and throwing error  
/\*Observable<String> values = Observable.error(new Exception("Oops"));  
Disposable disposable = values.subscribe(  
 v -> System.out.println("Received: " + v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
);\*/  
  
//defer sets logic to the class, in this case it will be calculating time after each subscribing  
//but if we use common "just" it will calculate time only once and return this value to each subscriber  
/\*Observable<Long> now = Observable.defer(() ->  
 Observable.just(System.currentTimeMillis()));  
  
now.subscribe(System.out::println);  
Thread.sleep(1000);  
now.subscribe(System.out::println);\*/  
  
//create returns observable with own circle of activities,  
// in that case it will use onNext and onComplete after each subscribing  
/\*Observable<String> values = Observable.create(o -> {  
 o.onNext("Hello");  
 o.onComplete();  
});  
Disposable disposable = values.subscribe(  
 v -> System.out.println("Received: " + v), // won't work because of create  
 e -> System.out.println("Error: " + e), // won't work because of create  
 () -> System.out.println("Completed") // won't work because of create  
);\*/  
  
  
  
  
  
//afrter subscribing will use this action to print values from 10 to 24 (10 - start, 15 - count)  
/\*Observable<Integer> values = Observable.range(10, 15);  
values.subscribe(System.out::println)\*/;  
  
//after subscribing will be calling onNext unlimited times or until we unsubscribe,  
// each recall will be separate from another by set time (1000 in example), exit: 1, 2, 3, 4, ...  
/\*Observable<Long> values = Observable.interval(1000, TimeUnit.MILLISECONDS);  
Disposable disposable = values.subscribe(  
 v -> System.out.println("Received: " + v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
);\*/  
  
/\*//will run method after period of time and call onComplete after one action  
Observable<Long> values = Observable.timer(1, TimeUnit.SECONDS);  
//will run method after period of time (2) and will be recalling it after second perod of time (1)  
Observable<Long> values = Observable.timer(2, 1, TimeUnit.SECONDS);  
Disposable disposable = values.subscribe(  
 v -> System.out.println("Received: " + v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
);\*/  
  
//it will run action and onCompleted after futureTask returns result, or throw exception of there is no result  
/\*FutureTask<Integer> f = new FutureTask<Integer>(() -> {  
 Thread.sleep(2000);  
 return 21;  
});  
new Thread(f).start();  
Observable<Integer> values = Observable.fromFuture(f); //wait for result  
Observable<Integer> values = Observable.from(f, 1000, TimeUnit.MILLISECONDS); // wai only a period of time  
Disposable disposable = values.subscribe(  
 v -> System.out.println("Received: " + v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
);\*/  
  
//run action with all methods from array and onCompleted at the end  
/\*Integer[] is = {1,2,3};  
Observable<Integer> values = Observable.fromArray(is);  
Disposable disposable = values.subscribe(  
 v -> System.out.println("Received: " + v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
);\*/

Filters

//adding filter  
/\*Observable<Integer> values = Observable.range(0,10);  
Disposable disposable = values  
 .filter(v -> v % 2 == 0)  
 .subscribe(  
 v -> System.out.println(v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
 );\*/  
  
  
//filter which deleted similar item if it already exists  
/\*Observable<Integer> values = Observable.create(o -> {  
 o.onNext(1);  
 o.onNext(1);  
 o.onNext(2);  
 o.onNext(3);  
 o.onNext(2);  
 o.onComplete();  
});  
Disposable disposable = values  
 .distinct() // this method works as filter  
 .subscribe(  
 v -> System.out.println(v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
 );\*/  
  
//filter by 1st char of text (deletes value with the same first char)  
/\*Observable<String> values = Observable.create(o -> {  
 o.onNext("First");  
 o.onNext("Second");  
 o.onNext("Third");  
 o.onNext("Fourth");  
 o.onNext("Fifth");  
 o.onComplete();  
});  
Disposable disposable = values  
 .distinct(v -> v.charAt(0)) // overriding method, setting key of sorting  
 .subscribe(  
 v -> System.out.println(v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
 );\*/  
  
// ignore all actions except onError and onComplete  
/\*Subscription subscription = values  
 .ignoreElements()  
 .subscribe(  
 v -> System.out.println(v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
 );\*/  
  
//can take first n elements or skip them, also can take elements for n time or skip them  
/\*Observable<Long> values = Observable.interval(100, TimeUnit.MILLISECONDS);  
Disposable disposable = values  
 .take(250, TimeUnit.MILLISECONDS) - take first 250 ms  
 .take(3) - take first 3  
 .takeWhile(v -> v < 2) - take while true  
 .skip(3) - skip first 3  
 .skip(250, TimeUnit.MILLISECONDS) - skip first 250 ms  
 .skipWhile(v -> v < 2) - skip while true  
 .subscribe(  
 v -> System.out.println(v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
 );\*/

Sorting data

//this method return true or false (in this case true)  
// because all checks if incoming values are true and method wil get only one argument rue or false  
/\*Observable<Integer> values = Observable.create(o -> {  
 o.onNext(0);  
 o.onNext(10);  
 o.onNext(10);  
 o.onNext(2);  
 o.onComplete();  
});  
Disposable disposable = values  
 .all(i -> i % 2 == 0)  
 .subscribe(  
 v -> System.out.println("" + v), // her we get true or false  
 e -> System.out.println("Error: " + e)  
 );\*/  
  
//get element at position  
/\*Observable<Integer> values = Observable.range(100, 10);  
Disposable disposable = values  
 .elementAt(2)  
 .subscribe(  
 v -> System.out.println(v),  
 e -> System.out.println("Error: " + e),  
 () -> System.out.println("Completed")  
 );\*/  
  
//check if 2 arrays ar equals are to each other  
/\*Observable<String> strings = Observable.just("1", "2", "3");  
Observable<Integer> ints = Observable.just(1, 2, 3);  
Observable.sequenceEqual(strings, ints, (s,i) -> s.equals(i.toString()))  
 .subscribe(  
 v -> System.out.println(v + ""),  
 e -> System.out.println("Error: " + e)  
 );\*/