Key Terminal Operators
in the Observable Class (Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Recognize key Observable operators
  - Concurrency & scheduler operators
  - Factory method operators
  - Action operators
  - Suppressing operators
  - Terminal operators
    - Terminate an Observable stream & trigger all the processing of operators in the stream
      - e.g., subscribe()

The subscribe() operator is non-blocking, unlike blockingSubscribe()
Key Terminal Operators in the Observable Class
Key Terminal Operators in the Observable Class

- The subscribe() operator
- Subscribe Consumers & a Runnable to this Observable

```java
Disposable subscribe
(Consumer<? super T> consumer,
 Consumer<? super Throwable> errorConsumer,
 Runnable completeConsumer)
```

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#subscribe](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#subscribe)
Key Terminal Operators in the Observable Class

- The `subscribe()` operator
- Subscribe Consumers & a Runnable to this Observable
- The params consume all elements in the sequence, handle errors, & react to completion

```java
Disposable subscribe(
Consumer<? super T> consumer,
Consumer<? super Throwable> errorConsumer,
Runnable completeConsumer)
```

**Interface Consumer<T>**

Type Parameters:
T - the type of the input to the operation

All Known Subinterfaces:
Stream.Builder<T>

Functional Interface:
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

Key Terminal Operators in the Observable Class

- The subscribe() operator
  - Subscribe Consumers & a Runnable to this Observable
  - The params consume all elements in the sequence, handle errors, & react to completion
    - This subscription requests unbounded demand
      - i.e., Long.MAX_VALUE

```java
Disposable subscribe
(Consumer<? super T> consumer,
 Consumer<? super Throwable>
 errorConsumer,
 Runnable completeConsumer)
```
Key Terminal Operators in the Observable Class

- The subscribe() operator
- Subscribe Consumers & a Runnable to this Observable
  - The params consume all elements in the sequence, handle errors, & react to completion
    - This subscription requests unbounded demand
- Signals emitted to this operator are represented by the following regular expression:
  
```java
subscribe

@CheckReturnValue
@SchedulerSupport(value="none")
@NonNull
public final @NonNull Disposable subscribe(@NonNull @NonNull Consumer<? super T> onNext,
          @NonNull @NonNull Consumer<? super Throwable> onError,
          @NonNull @NonNull Action onComplete)

Subscribes to the current Observable and provides callbacks to handle the items it emits and any error or completion notification it signals.

Scheduler:
  subscribe does not operate by default on a particular Scheduler.

Parameters:
  onNext - the Consumer<T> you have designed to accept emissions from the current Observable
  onError - the Consumer<Throwable> you have designed to accept any error notification from the current Observable
  onComplete - the Action you have designed to accept a completion notification from the current Observable

Returns:
  the new Disposable instance that can be used to dispose the subscription at any time
```

`onNext()*(onComplete()|onError())?`
Key Terminal Operators in the Observable Class

- The `subscribe()` operator
- Subscribe Consumers & a Runnable to this Observable
  - The params consume all elements in the sequence, handle errors, & react to completion
  - A Disposable is returned, which indicates a task or resource that can be cancelled/disposed

```java
Disposable subscribe
(Consumer<? super T> consumer,
 Consumer<? super Throwable> errorConsumer,
 Runnable completeConsumer)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/disposables/Disposable.html
• The subscribe() operator
  • Subscribe Consumers & a Runnable to this Observable
    • The params consume all elements in the sequence, handle errors, & react to completion
  • A Disposable is returned, which indicates a task or resource that can be cancelled/disposed
    • Disposables can be accumulated & disposed in one fell swoop!

CompositeDisposable
mDisposables
  (mPublisherScheduler,
mSubscriberScheduler,
mSubscriber);

... mDisposables.dispose();

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/disposables/CompositeDisposable.html
Key Terminal Operators in the Observable Class

- The subscribe() operator
- Subscribe Consumers & a Runnable to this Observable
- This operator triggers all the processing in a chain

```java
Observable
   .fromIterable
   (bigFractionList)
   .map(fraction -> fraction
       .multiply(sBigReducedFraction))
   .subscribe(fraction -> sb
       .append(" = "+ fraction
       .toMixedString()
       + "\n"),
   error -> { sb
       .append("error"); ... },
   () -> BigFractionUtils
   .display(sb.toString()));
```

See Reactive/Observable/ex2/src/main/java/ObservableEx.java
Key Terminal Operators in the Observable Class

- The subscribe() operator
  - Subscribe Consumers & a Runnable to this Observable
  - This operator triggers all the processing in a chain

```java
Observable
   .fromIterable
   (bigFractionList)
   .map(fraction -> fraction
       .multiply(sBigReducedFraction))
   .subscribe(fraction -> sb
       .append(" = "+
       + fraction
       .toMixedString()
       + "\n"),
   error -> { sb
       .append("error"); ... }
},
() -> BigFractionUtils.display(sb.toString()));
```
Key Terminal Operators in the Observable Class

- The subscribe() operator
- Subscribe Consumers & a Runnable to this Observable
- This operator triggers all the processing in a chain

```java
Observable
  .fromIterable
    (bigFractionList)
  .map(fraction -> fraction
    .multiply(sBigReducedFraction))
  .subscribe(fraction -> sb
    .append(" = "+ fraction
    .toString() + "\n"),
  error -> { sb
    .append("error"); ...}
) -> BigFractionUtils
  .display(sb.toString());
```

Error Processing
Key Terminal Operators in the Observable Class

- The subscribe() operator
  - Subscribe Consumers & a Runnable to this Observable
- This operator triggers all the processing in a chain

```java
Observable
  .fromIterable
    (bigFractionList)
  .map(fraction -> fraction
    .multiply(sBigReducedFraction))
  .subscribe(fraction -> sb
    .append(" = "+ fraction
    .toString()
    + 
    + "\n"),
  error -> { sb
    .append("error"); ...
  },
  () -> BigFractionUtils
    .display(sb.toString()));
```

Completion Processing
Key Terminal Operators in the Observable Class

• The subscribe() operator
  • Subscribe Consumers & a Runnable to this Observable
  • This operator triggers all the processing in a chain
  • Calling this operator will not block the caller thread
  • Until upstream terminates normally or with an error

```
subscribe

@CheckReturnValue
@SchedulerSupport(value="none")
public final @NonNull Disposable subscribe(@NonNull Consumer<? super T> onNext,
                                          @NonNull Consumer<? super Throwable> onError)

Subscribes to the current Observable and provides callbacks to handle the items it emits and any error notification it signals.

Scheduler:
subscribe does not operate by default on a particular Scheduler.

Parameters:
onNext - the Consumer<T> you have designed to accept emissions from the current Observable
onError - the Consumer<Throwable> you have designed to accept notification from the current Observable

Returns:
the new Disposable instance that can be used to dispose the subscription at a time

Throws:
NullPointerException - if onNext or onError is null

See Also:
ReactiveX operators documentation: Subscribe
```
Key Terminal Operators in the Observable Class

• The subscribe() operator
  • Subscribe Consumers & a Runnable to this Observable
  • This operator triggers all the processing in a chain
  • Calling this operator will not block the caller thread
    • Until upstream terminates normally or with an error
  • These semantics motivate the need for the AsyncTaskBarrier framework!

See Reactive/Single/ex1/src/main/java/utils/AsyncTaskBarrier.java
Key Terminal Operators in the Observable Class

- The `subscribe()` operator
  - Subscribe Consumers & a Runnable to this Observable
  - This operator triggers all the processing in a chain
  - Calling this operator will *not* block the caller thread
  - Other versions of `subscribe()` support different capabilities

```java
void subscribe (Observer<? super T> observer)
```

Subscribes the given Observer to this `ObservableSource` instance, which provides additional capabilities for receiving push-based notifications

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#subscribe](http://reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#subscribe)
Key Terminal Operators in the Observable Class

- The subscribe() operator
  - Subscribe Consumers & a Runnable to this Observable
  - This operator triggers all the processing in a chain
  - Calling this operator will *not* block the caller thread
  - Other versions of subscribe() support different capabilities
- Project Reactor’s operator Flux .subscribe() works the same

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#subscribe](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#subscribe)
End of Key Terminal Operators in the Observable Class (Part 2)