Key Error Handling Operators in the Observable Class

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
  - Factory method operators
  - Transforming operators
  - Concurrency & scheduler operators
  - Error handling operators
    - These operators handle errors that occur in an Observable chain
      - e.g., onErrorReturn() & onErrorResumeNext()

Key Error Handling Operators in the Observable Class

Key Error Handling Operators in the Observable Class

- The onErrorReturn() operator
  - Ends the flow with a last item returned by a Function

```
Observable<T>
onErrorReturn
   (Function<? super Throwable,
                ? extends Publisher
                     <? extends T>>
    itemSupplier)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#onErrorReturn

Key Error Handling Operators in the Observable Class

- The onErrorReturn() operator
  - Ends the flow with a last item returned by a Function
    - The Function param returns one value that will be emitted along with a regular onComplete() if the current Observable signals an onError() event

```
Observable<T>
onErrorReturn
   (Function<? super Throwable,
             ? extends Publisher
                    <? extends T>>
    itemSupplier)
```

Interface Function<T,R>

Type Parameters:

T - the type of the input to the function

R - the type of the result of the function

All Known Subinterfaces:

UnaryOperator<T>

Functional Interface:

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

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Function.html

Key Error Handling Operators in the Observable Class

- The onErrorReturn() operator
  - Ends the flow with a last item returned by a Function
    - The Function param returns one value that will be emitted along with a regular onComplete() if the current Observable signals an onError() event
      - The type of the error is a subclass of Throwable

```
Observable<T>
onErrorReturn
   (Function<? super Throwable,
              ? extends Publisher
                   <? extends T>>
    itemSupplier)
```

```
public class Throwable
extends Object
implements Serializable
```

The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement. Similarly, only this class or one of its subclasses can be the argument type in a catch clause. For the purposes of compile-time checking of exceptions, Throwable and any subclass of Throwable that is not also a subclass of either RuntimeException or Error are regarded as checked exceptions.

See docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html

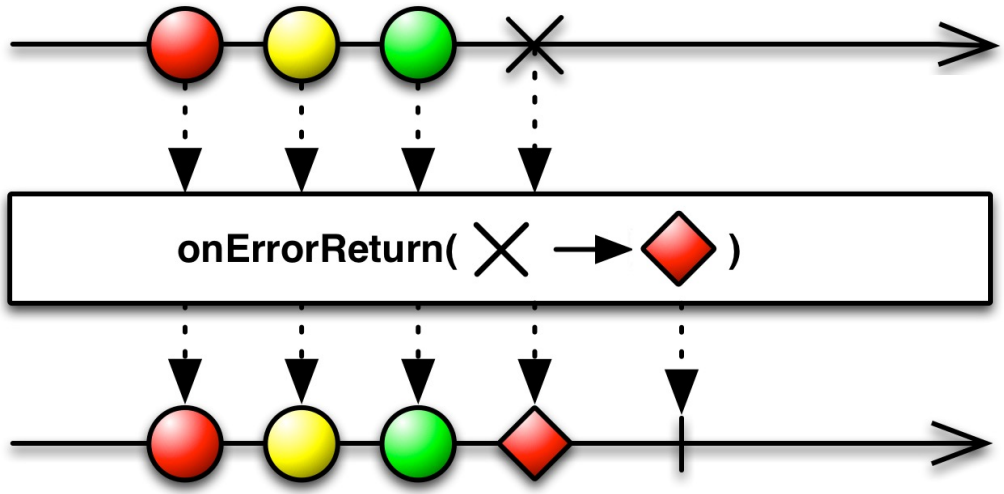
Key Error Handling Operators in the Observable Class

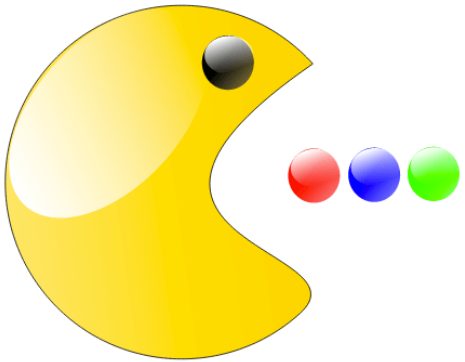
- The onErrorReturn() operator
  - Ends the flow with a last item returned by a Function
    - The Function param returns one value that will be emitted along with a regular onComplete() if the current Observable signals an onError() event
  - Returns a new Observable that falls back upon itemSupplier on an error

```
Observable<T>
onErrorReturn
   (Function<? super Throwable,
               ? extends Publisher
                     <? extends T>>
    itemSupplier)
```

Key Error Handling Operators in the Observable Class

- The onErrorReturn() operator
  - Ends the flow with a last item returned by a Function
  - This operator "swallows" the exception so it won't propagate up the call chain/stack further

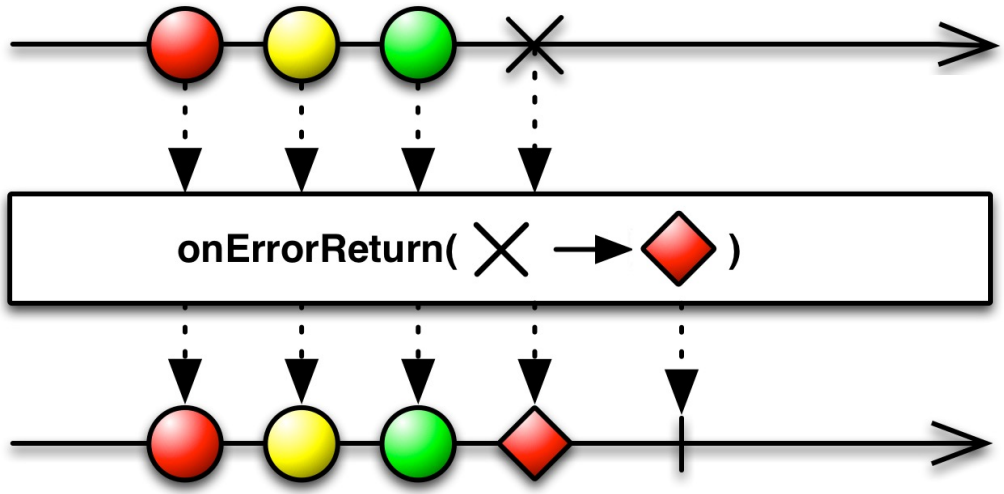




See en.wikipedia.org/wiki/Error_hiding

Key Error Handling Operators in the Observable Class

- The onErrorReturn() operator
  - Ends the flow with a last item returned by a Function
  - This operator "swallows" the exception so it won't propagate up the call chain/stack further



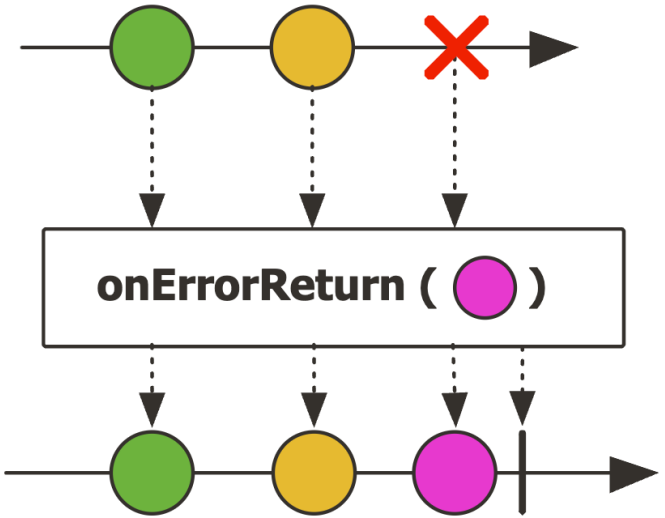

```
return Observable
   .fromCallable(BigFraction
                    .valueOf(Math.abs(sRANDOM.nextInt()),
                             denominator))
   .subscribeOn(Schedulers.computation())
   .onErrorReturn(errorHandler)
   .map(multiplyBigFractions);
```

Convert ArithmeticException to 0 when denominator == 0

See Reactive/observable/ex3/src/main/java/ObservableEx.java

Key Error Handling Operators in the Observable Class

- The onErrorReturn() operator
  - Ends the flow with a last item returned by a Function
  - This operator "swallows" the exception so it won't propagate up the call chain/stack further
  - Project Reactor's operator Flux .onErrorReturn() works the same



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#onErrorReturn

Key Error Handling Operators in the Observable Class

- The onErrorReturn() operator
  - Ends the flow with a last item returned by a Function
  - This operator "swallows" the exception so it won't propagate up the call chain/stack further
  - Project Reactor's operator Flux .onErrorReturn() works the same
  - The Java CompletableFuture exceptionally() method is similar

exceptionally

```
CompletionStage<T> exceptionally(
Function<Throwable,? extends T> fn)
```

Returns a new CompletionStage that, when this stage completes exceptionally, is executed with this stage's exception as the argument to the supplied function. Otherwise, if this stage completes normally, then the returned stage also completes normally with the same value.

Parameters:

`fn` - the function to use to compute the value of the returned CompletionStage if this CompletionStage completed exceptionally

Returns:

the new CompletionStage

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#exceptionally

Key Error Handling Operators in the Observable Class

- The onErrorResumeNext() operator
  - Resumes the flow instead of signaling an error via onError()

```
Observable<T> onErrorResumeNext
  (Function<? super Throwable,
      ? extends ObservableSource
                    <? extends T>>
   fallbackSupplier)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#onErrorResumeNext

Key Error Handling Operators in the Observable Class

- The onErrorResumeNext() operator
  - Resumes the flow instead of signaling an error via onError()
    - The param is a Function that chooses the fallback, depending on the type of the error

```
Observable<T> onErrorResumeNext
  (Function<? super Throwable,
      ? extends ObservableSource
                    <? extends T>>
   fallbackSupplier)
```

Interface Function<T,R>

Type Parameters:

T - the type of the input to the function

R - the type of the result of the function

All Known Subinterfaces:

UnaryOperator<T>

Functional Interface:

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

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Function.html

Key Error Handling Operators in the Observable Class

- The onErrorResumeNext() operator
  - Resumes the flow instead of signaling an error via onError()
    - The param is a Function that chooses the fallback, depending on the type of the error
      - The type of the error is a subclass of Throwable

```
Observable<T> onErrorResumeNext
  (Function<? super Throwable,
      ? extends ObservableSource
                  <? extends T>>
   fallbackSupplier)
```

```
public class Throwable
extends Object
implements Serializable
```

The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement. Similarly, only this class or one of its subclasses can be the argument type in a catch clause. For the purposes of compile-time checking of exceptions, Throwable and any subclass of Throwable that is not also a subclass of either RuntimeException or Error are regarded as checked exceptions.

See docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html

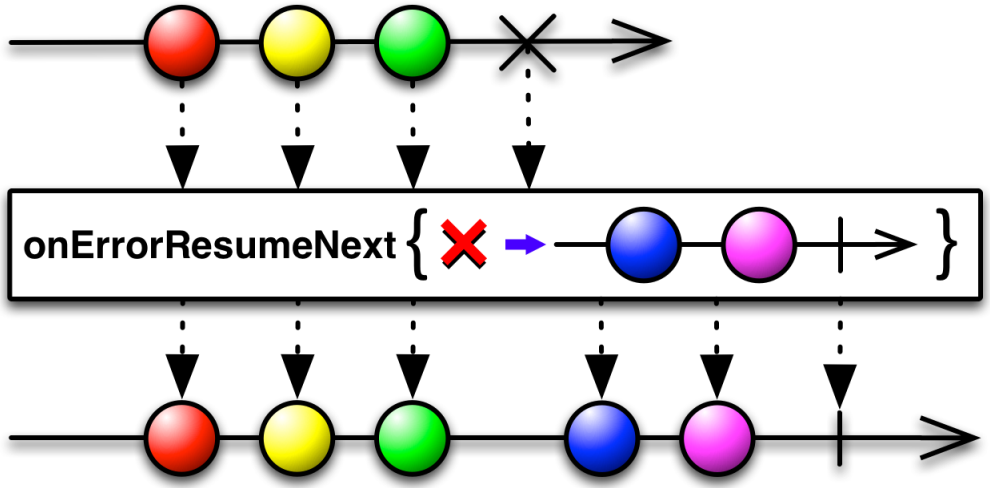
Key Error Handling Operators in the Observable Class

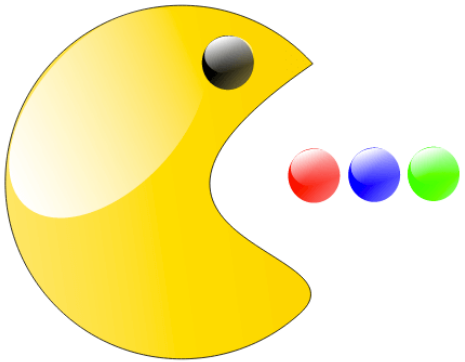
- The onErrorResumeNext() operator
  - Resumes the flow instead of signaling an error via onError()
    - The param is a Function that chooses the fallback, depending on the type of the error
    - Returns an Observable that returns an ObservableSource that will take over if the current Observable encounters an error

```
Observable<T> onErrorResumeNext
  (Function<? super Throwable,
      ? extends ObservableSource
                    <? extends T>>
   fallbackSupplier)
```

Key Error Handling Operators in the Observable Class

- The onErrorResumeNext() operator
  - Subscribe to a returned fallback publisher when any error occurs
  - This operator "swallows" the exception so it won't propagate up the call chain/stack further

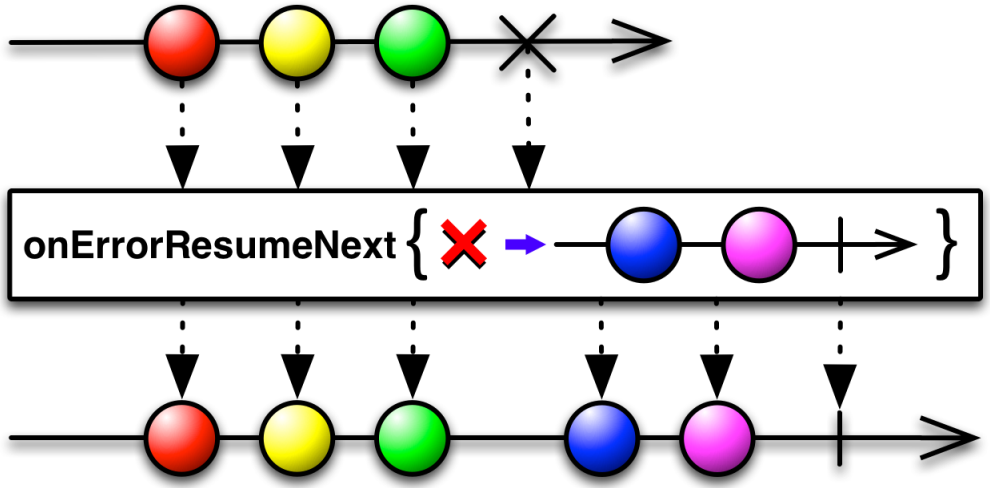




See en.wikipedia.org/wiki/Error_hiding

Key Error Handling Operators in the Observable Class

- The onErrorResumeNext() operator
  - Subscribe to a returned fallback publisher when any error occurs
  - This operator "swallows" the exception so it won't propagate up the call chain/stack further



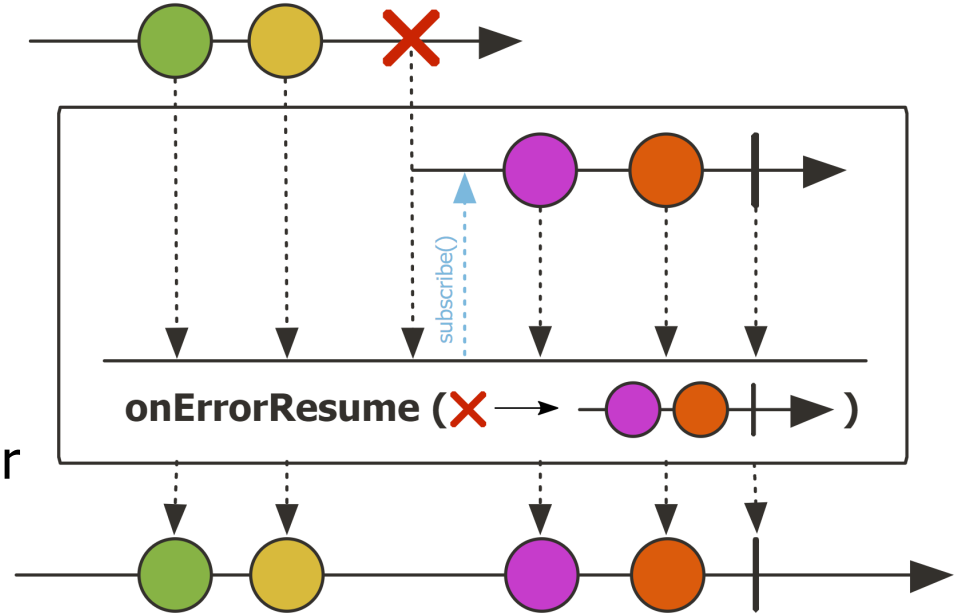

```
return Observable
   .fromIterable(denominators)
   .map(denominator -> BigFraction
        .valueOf(Math.abs(sRANDOM.nextInt()), denominator))
   .onErrorResumeNext(__ -> Observable.empty())
   .collect(toList())
   ...
```

Convert the thrown Arithmetic Exception to empty Observable

See Reactive/observable/ex3/src/main/java/ObservableEx.java

Key Error Handling Operators in the Observable Class

- The onErrorResumeNext() operator
  - Subscribe to a returned fallback publisher when any error occurs
  - This operator "swallows" the exception so it won't propagate up the call chain/stack further
  - The Flux.onErrorResume() operator in Project Reactor works the same



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#onErrorResume

Key Error Handling Operators in the Observable Class

- The onErrorResumeNext() operator
  - Subscribe to a returned fallback publisher when any error occurs
  - This operator "swallows" the exception so it won't propagate up the call chain/stack further
  - The Flux.onErrorResume() operator in Project Reactor works the same
  - The Java CompletableFuture exceptionally() method is similar

exceptionally

```
CompletionStage<T> exceptionally(
Function<Throwable,? extends T> fn)
```

Returns a new CompletionStage that, when this stage completes exceptionally, is executed with this stage's exception as the argument to the supplied function. Otherwise, if this stage completes normally, then the returned stage also completes normally with the same value.

Parameters:

`fn` - the function to use to compute the value of the returned CompletionStage if this CompletionStage completed exceptionally

Returns:

the new CompletionStage

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#exceptionally

End of Key Error Handling Operators in the Observable Class