An Observable typically does not *throw* exceptions. Instead it notifies any observers that an unrecoverable error has occurred by terminating the Observable sequence with an `onError` notification.

There are some exceptions to this. For example, if the `onError()` call *itself* fails, the Observable will not attempt to notify the observer of this by again calling `onError` but will throw a `RuntimeException`, an `OnErrorFailedException`, or an `OnErrorNotImplementedException`.

Techniques for recovering from onError notifications

So rather than *catch* exceptions, your observer or operator should more typically respond to `onError` notifications of exceptions. There are also a variety of Observable operators that you can use to react to or recover from `onError` notifications from Observables. For example, you might use an operator to:

1. swallow the error and switch over to a backup Observable to continue the sequence
2. swallow the error and emit a default item
3. swallow the error and immediately try to restart the failed Observable
4. swallow the error and try to restart the failed Observable after some back-off interval

You can use the operators described in [[Error Handling Operators]] to implement these strategies.

RxJava-specific exceptions and what to do about them

CompositeException

This indicates that more than one exception occurred. You can use the exception's getExceptions() method to retrieve the individual exceptions that make up the composite.

MissingBackpressureException

This indicates that a Subscriber or operator attempted to apply reactive pull backpressure to an Observable that does not implement it. See [[Backpressure]] for work-arounds for Observables that do not implement reactive pull backpressure.

OnErrorFailedException

This indicates that an Observable tried to call its observer's onError() method, but that method itself threw an exception.

OnErrorNotImplementedException

This indicates that an Observable tried to call its observer's onError() method, but that no such method existed. You can eliminate this by either fixing the Observable so that it no longer reaches an error condition, by implementing an onError handler in the observer, or by intercepting the onError notification before it reaches the observer by using one of the operators described elsewhere on this page.

OnErrorThrowable

Observers pass throwables of this sort into their observers' onError() handlers. A Throwable of this variety contains more information about the error and about the Observable-specific state of the system at the time of the error than does a standard Throwable.