Applying Key Methods in the Single Class (Part 1)

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 methods in the Single class & how they are applied in the case studies

Class Single<T>

java.lang.Object
io.reactivex.rxjava3.core.Single<T>

Type Parameters:
T - the type of the item emitted by the `Single`

All Implemented Interfaces:
SingleSource<T>

Direct Known Subclasses:
SingleSubject

```
public abstract class Single<T>
extends Object
implements SingleSource<T>
```

The `Single` class implements the Reactive Pattern for a single value response.

`Single` behaves similarly to `Observable` except that it can only emit either a single successful value or an error (there is no `onComplete` notification as there is for an `Observable`).

The `Single` class implements the `SingleSource` base interface and the default consumer type it interacts with is the `SingleObserver` via the `subscribe(SingleObserver)` method.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html

Learning Objectives in this Part of the Lesson

- Recognize key methods in the Single class & how they are applied in the case studies
  - Case study ex1

```
return Single
    .fromCallable(reduceFraction)

    .map(convertToMixedString)

    .doOnSuccess(printResult)

    .ignoreElement();
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Single/ex1

Applying Key Methods in the Single Class to ex1

Applying Key Methods in the Single Class to ex1

- ex1 shows how to apply RxJava features *synchronously* to perform basic Single operations
  - e.g., fromCallable(), doOnSuccess(), ignoreElement(), & map()

```
return Single
    .fromCallable(reduceFraction)

    .map(convertToMixedString)

    .doOnSuccess(printResult)

    .ignoreElement();
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Single/ex1

Applying Key Methods in the Single Class to ex1

- The fromCallable() method
  - This factory method creates & returns a Single of type T

```
static <T> Single<T> fromCallable
   (Callable<? extends T> supplier)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#fromCallable

Applying Key Methods in the Single Class to ex1

- The fromCallable() method
  - This factory method creates & returns a Single of type T
    - The Single's value is produced via the provided Callable supplier

```
static <T> Single<T> fromCallable
    (Callable<? extends T> supplier)
```

Interface Callable<V>

Type Parameters:

V - the result type of method call

All Known Subinterfaces:

DocumentationTool.DocumentationTask, JavaCompiler.CompilationTask

Functional Interface:

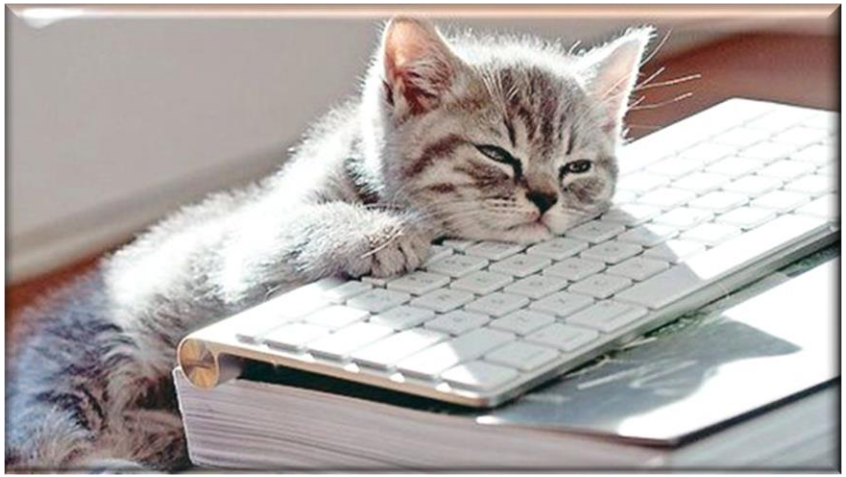
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html

Applying Key Methods in the Single Class to ex1

- The fromCallable() method
  - This factory method creates & returns a Single of type T
    - The Single's value is produced via the provided Callable supplier
      - The callable is invoked at the time of subscription & also for each subscriber
        - i.e., it's "lazy"

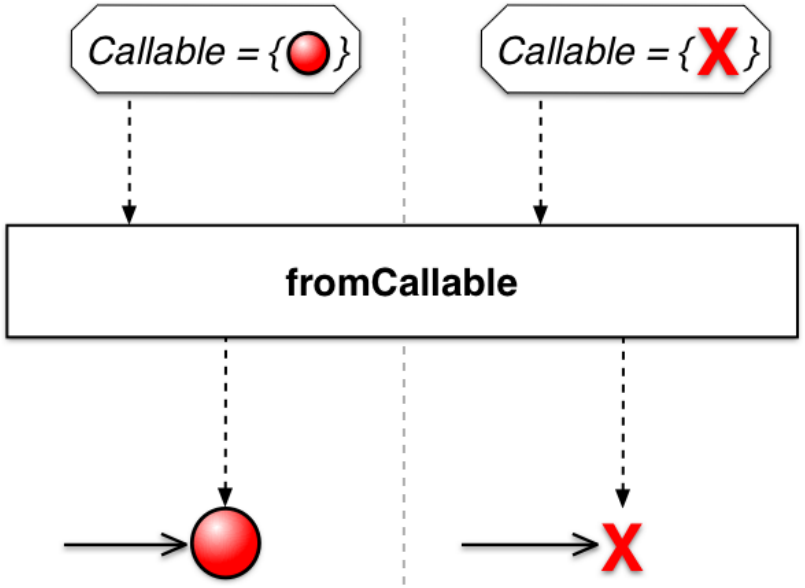
```
static <T> Single<T> fromCallable
    (Callable<? extends T> supplier)
```



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html

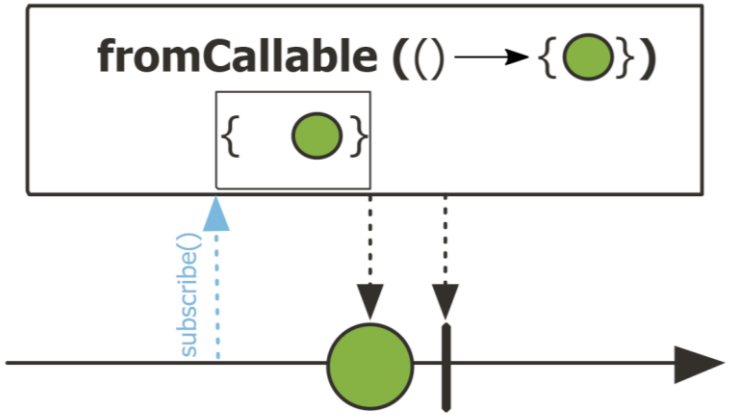
Applying Key Methods in the Single Class to ex1

- The fromCallable() method
  - This factory method creates & returns a Single of type T
  - This factory method adapts non-reactive input sources into the reactive model



Applying Key Methods in the Single Class to ex1

- The fromCallable() method
  - This factory method creates & returns a Single of type T
  - This factory method adapts non-reactive input sources into the reactive model
  - Project Reactor's Mono.fromCallable() method works the same way



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#fromCallable

Applying Key Methods in the Single Class to ex1

- The map() method
  - Transform the item emitted by this Single

```
<R> Single<R>
map(Function<? super T,? extends R>
        mapper)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#map

Applying Key Methods in the Single Class to ex1

- The map() method
  - Transform the item emitted by this Single
    - Applies a synchronous function to transform the item

```
<R> Single<R>
map(Function<? super T,? extends R>
        mapper)
```

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>

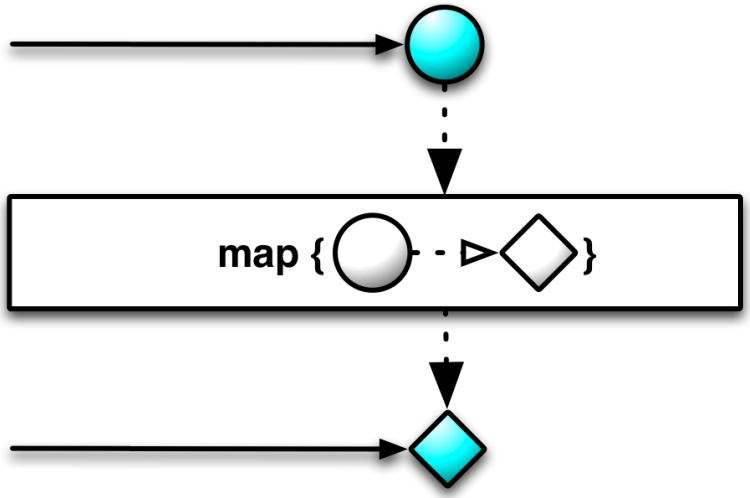
Functional Interface:

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

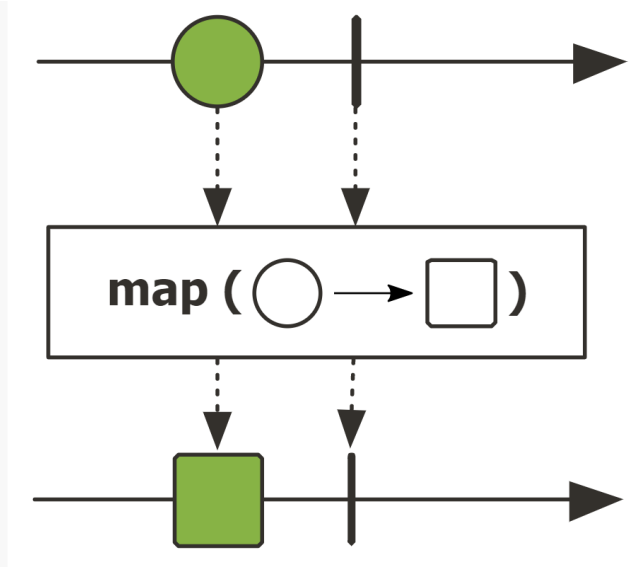
Applying Key Methods in the Single Class to ex1

- The map() method
  - Transform the item emitted by this Single
    - Applies a synchronous function to transform the item
    - map() can transform the type of elements it processes



Applying Key Methods in the Single Class to ex1

- The map() method
  - Transform the item emitted by this Single
  - Project Reactor's Mono.map() method works the same way



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Single.html#map

Applying Key Methods in the Single Class to ex1

- The map() method
  - Transform the item emitted by this Single
  - Project Reactor's Mono.map() method works the same way
  - Similar to Java Completable Future.thenApply() method

thenApply

```
public <U> CompletableFuture<U> thenApply(Function<? super T,? extends U> fn)
```

Description copied from interface: CompletionStage

Returns a new CompletionStage that, when this stage completes normally, is executed with this stage's result as the argument to the supplied function. See the CompletionStage documentation for rules covering exceptional completion.

Specified by:

thenApply in interface CompletionStage<T>

Type Parameters:

U - the function's return type

Parameters:

fn - the function to use to compute the value of the returned CompletionStage

Returns:

the new CompletionStage

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

Applying Key Methods in the Single Class to ex1

- The doOnSuccess() method
  - Add a behavior triggered when the Single completes successfully

```
Single<T> doOnSuccess
   (Consumer<? super T>
    onSuccess)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#doOnSuccess

Applying Key Methods in the Single Class to ex1

- The doOnSuccess() method
  - Add a behavior triggered when the Single completes successfully
    - The behavior is passed as a consumer param that's called on successful completion

```
Single<T> doOnSuccess
   (Consumer<? super T>
    onSuccess)
```

Interface Consumer<T>

Type Parameters:

T - the type of the input to the operation

All Known Subinterfaces:

Stream.Builder<T>

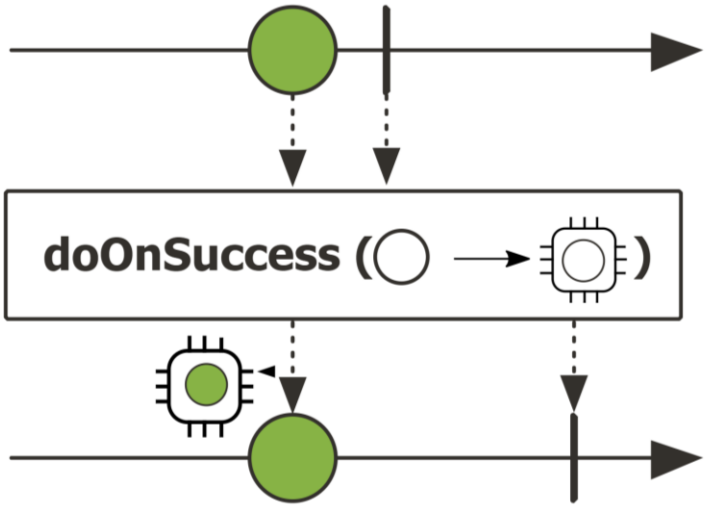
Functional Interface:

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

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

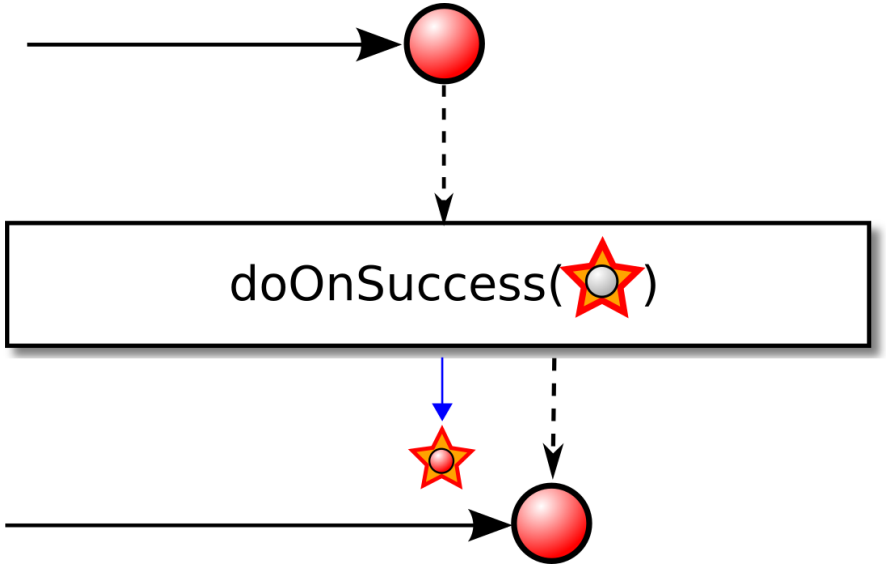
Applying Key Methods in the Single Class to ex1

- The doOnSuccess() method
  - Add a behavior triggered when the Single completes successfully
  - The actual value emitted by doOnSuccess() is not modified



Applying Key Methods in the Single Class to ex1

- The doOnSuccess() method
  - Add a behavior triggered when the Single completes successfully
  - The actual value emitted by doOnSuccess() is not modified
  - Project Reactor's method Mono.doOnSuccess() works the same way



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Single.html#doOnSuccess

Applying Key Methods in the Single Class to ex1

- The doOnSuccess() method
  - Add a behavior triggered when the Single completes successfully
  - The actual value emitted by doOnSuccess() is not modified
  - Project Reactor's method Mono.doOnSuccess() works the same way
  - Similar to the Java Completable Future.thenAccept() method

thenAccept

```
public CompletableFuture<Void> thenAccept(Consumer<? super T> action)
```

Description copied from interface: CompletionStage

Returns a new CompletionStage that, when this stage completes normally, is executed with this stage's result as the argument to the supplied action. See the CompletionStage documentation for rules covering exceptional completion.

Specified by:

thenAccept in interface CompletionStage<T>

Parameters:

action - the action to perform before completing the returned CompletionStage

Returns:

the new CompletionStage

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

Applying Key Methods in the Single Class to ex1

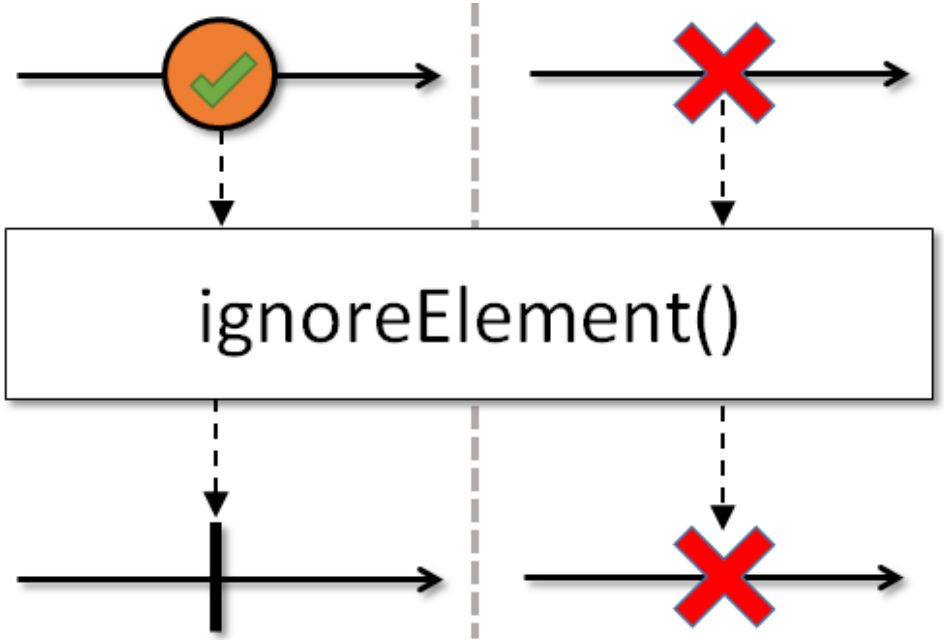
- The ignoreElement() method
  - Returns a Completable that ignores the success value of this Single & signals onComplete() instead

```
Completable ignoreElement()
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#ignoreElement

Applying Key Methods in the Single Class to ex1

- The ignoreElement() method
  - Returns a Completable that ignores the success value of this Single & signals onComplete() instead
  - This "data-suppressing" operator ignores its payload
    - It can be used to indicate when an async operation completes



Applying Key Methods in the Single Class to ex1

- The ignoreElement() method
  - Returns a Completable that ignores the success value of this Single & signals onComplete() instead
  - This "data-suppressing" operator ignores its payload
  - ignoreElement() returns a Completable value
    - Completable represents a deferred computation without any value, but only indicates completion or exception

Class Completable

java.lang.Object
io.reactivex.rxjava3.core.Completable

All Implemented Interfaces:
CompletableSource

Direct Known Subclasses:
CompletableSubject

```
public abstract class Completable
extends Object
implements CompletableSource
```

The `Completable` class represents a deferred computation without any value but only indication for completion or exception.

`Completable` behaves similarly to `Observable` except that it can only emit either a completion or error signal (there is no `onNext` or `onSuccess` as with the other reactive types).

The `Completable` class implements the `CompletableSource` base interface and the default consumer type it interacts with is the `CompletableObserver` via the `subscribe(CompletableObserver)` method. The `Completable` operates with the following sequential protocol:

```
onSubscribe (onError | onComplete)?
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Completable.html

Applying Key Methods in the Single Class to ex1

- The ignoreElement() method
  - Returns a Completable that ignores the success value of this Single & signals onComplete() instead
  - This "data-suppressing" operator ignores its payload
  - ignoreElement() returns a Completable value
  - ignoreElement() is needed for the AsyncTester framework
    - Ensures an async computation doesn't complete prematurely

Class AsyncTester

java.lang.Object
utils.AsyncTester

```
public class AsyncTester
extends java.lang.Object
```

This class asynchronously runs tests that use the RxJava framework and ensures that the test driver doesn't exit until all the asynchronous processing is completed.

Method Summary

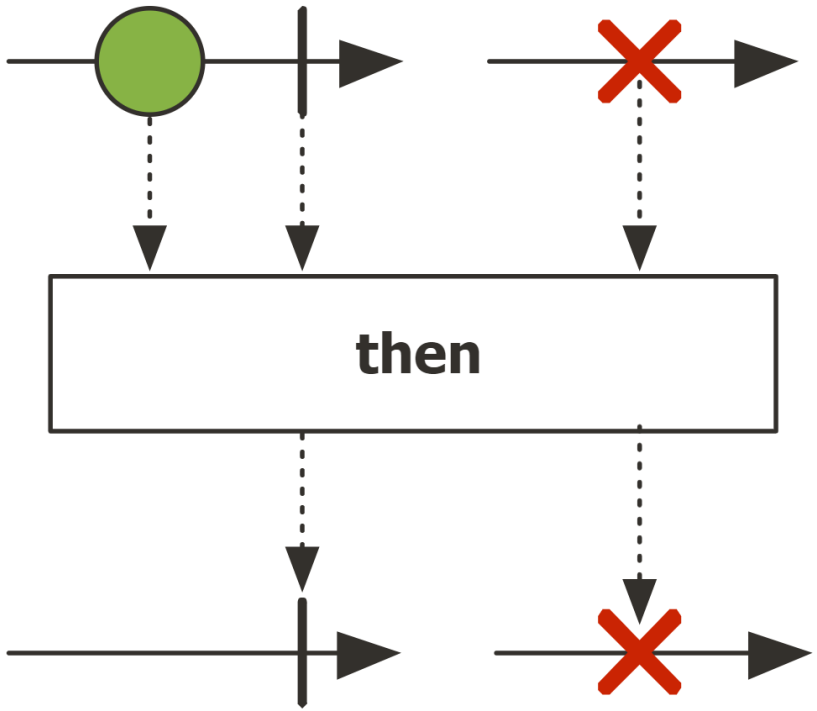
All Methods | Static Methods | Concrete Methods

Modifier and Type	Method
static void	register (io.reactivex.rxjava3.
static io.reactivex.rxjava3.core.Single<java.lang.Long>	runTests()

See Reactive/Single/ex1/src/main/java/utils/AsyncTester.java

Applying Key Methods in the Single Class to ex1

- The ignoreElement() method
  - Returns a Completable that ignores the success value of this Single & signals onComplete() instead
  - This "data-suppressing" operator ignores its payload
  - ignoreElement() returns a Completable value
  - ignoreElement() is needed for the AsyncTester framework
  - Project Reactor's Mono.then() method works in a similar way



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Single.html#then

Applying Key Methods in the Single Class to ex1

```java
/**
 * This class shows how to apply Project Reactor features
 * synchronously to to reduce and display BigFractions via
 * basic Mono operations, including fromCallable(), map(),
 * doOnSuccess(), and then().
 */
public class SingleEx {
    /**
     * Test synchronous BigFraction reduction using a mono and a
     * pipeline of operations that run on the calling thread.
     */
    public static Completable testFractionReductionSync1() {
        StringBuilder sb =
            new StringBuilder(">> Calling testFractionReductionSync1()\n");

        // Create a new unreduced big fraction.
        BigFraction unreducedFraction = BigFraction
            .valueOf(new BigInteger(sBI1),
                     new BigInteger(sBI2),
                     reduce: false);
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Single/ex1

End of Applying Key Methods in the Single Class (Part 1)