

Applying Key Methods in the Single Class

(Part 3)

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

- Case study ex1
- Case study ex2
- Case study ex3

```
return Single
    .just(BigFractionUtils
        .makeBigFraction(... )
        .multiply(sBigReducedFrac))

    .subscribeOn
        (Schedulers.parallel());
```

```
Random random = new Random();
```

```
Single<BigFraction> m1 =
    makeBigFraction(random);
Single<BigFraction> m2 =
    makeBigFraction(random);
```

```
return m1
    .zipWith(m2,
        BigFraction::add)

    .doOnSuccess
        (mixedFractionPrinter)

    .then();
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/Reactive/Single/ex3

Applying Key Methods in the Single Class in ex3

Applying Key Methods in the Single Class in ex3

- ex3 shows how to apply RxJava features asynchronously to perform various Single operations
- e.g., `subscribeOn()`, `doOnSuccess()`, `ignoreElement()`, `just()`, `zipWith()`, & `Schedulers.computation()`

```
return Single
    .just(BigFractionUtils
        .makeBigFraction(...)
        .multiply(sReducedFrac))
    .doOnSuccess(fractionPrinter)
    .subscribeOn
        (Schedulers.computation());
```

```
Random random = new Random();
```

```
Single<BigFraction> m1 =
    makeBigFraction(random);
Single<BigFraction> m2 =
    makeBigFraction(random);
```

```
return m1
    .zipWith(m2,
        BigFraction::add)
    .doOnSuccess
        (mixedFractionPrinter)
    .ignoreElement();
```

Applying Key Methods in the Single Class in ex3

- The just() method
 - Create a new Single that emits the specified item

```
static <T> Single<T> just(T data)
```

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

Applying Key Methods in the Single Class in ex3

- The just() method
 - Create a new Single that emits the specified item
 - This value is captured at instantiation time & is the value returned for all subscribers
 - i.e., it's "eager"

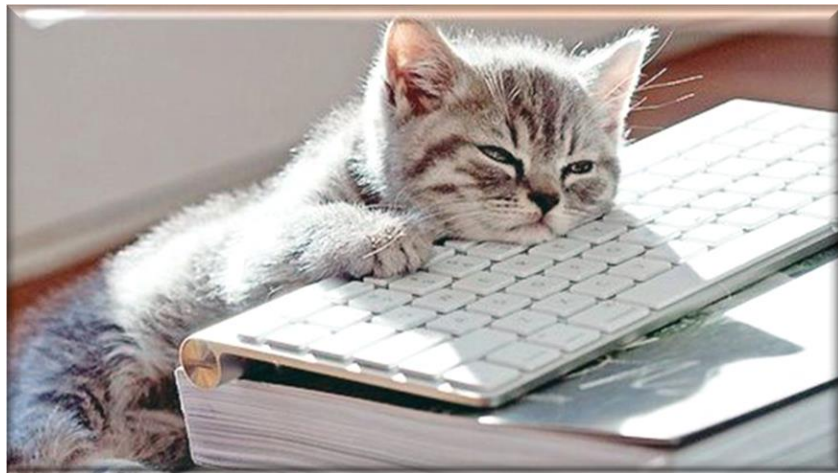
```
static <T> Single<T> just(T data)
```



Applying Key Methods in the Single Class in ex3

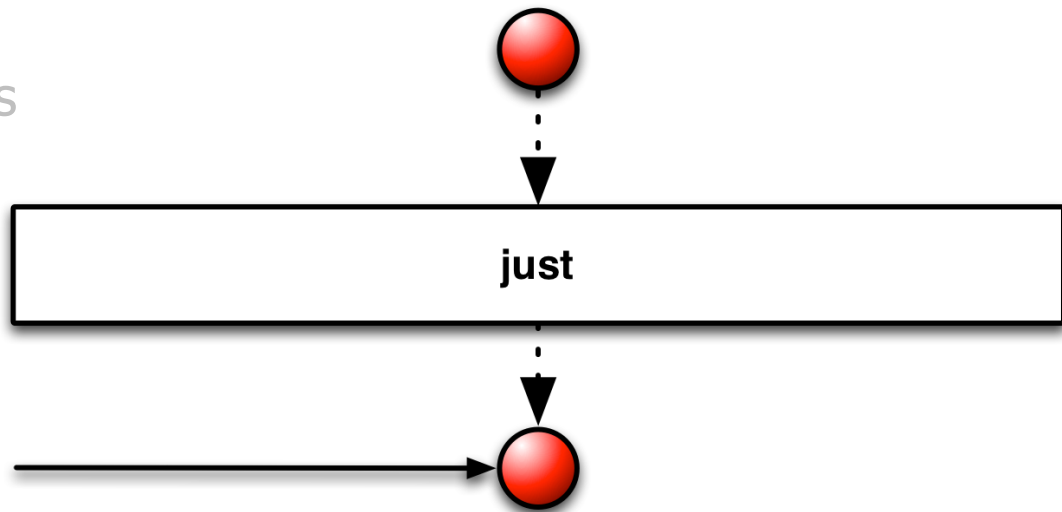
- The just() method
 - Create a new Single that emits the specified item
 - This value is captured at instantiation time & is the value returned for all subscribers
 - In contrast, Single.fromCallable() invokes the callable param at the time of subscription & separately for each subscriber
 - i.e., it's "lazy"

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



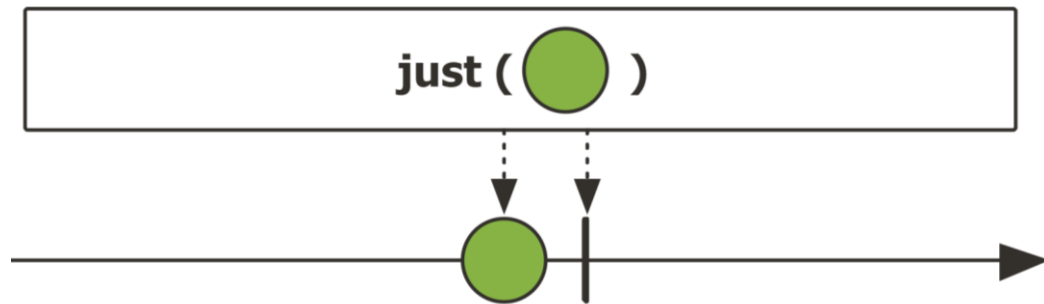
Applying Key Methods in the Single Class in ex3

- The just() method
 - Create a new Single that emits the specified item
 - This factory method adapts non-reactive input sources into the reactive model



Applying Key Methods in the Single Class in ex3

- The just() method
 - Create a new Single that emits the specified item
 - This factory method adapts non-reactive input sources into the reactive model
- Project Reactor's Mono.just() works the same way



Applying Key Methods in the Single Class in ex3

- The zipWith() method
 - Joins two results into a single result after they both emit

```
<T2, O> Single<O>
```

```
zipWith(Single<? extends T2> other,  
         BiFunction<? super T,  
                    ? super T2,  
                    ? extends O>  
         combinator)
```

Applying Key Methods in the Single Class in ex3

- The zipWith() method
 - Joins two results into a single result after they both emit
 - Combine the result from this & other Single into another object via a given combinator bifunction

```
<T2, O> Single<O>  
zipWith(Single<? extends T2> other,  
        BiFunction<? super T,  
                  ? super T2,  
                  ? extends O>  
        combinator)
```

Interface BiFunction<T1,T2,R>

Type Parameters:

T1 - the first value type

T2 - the second value type

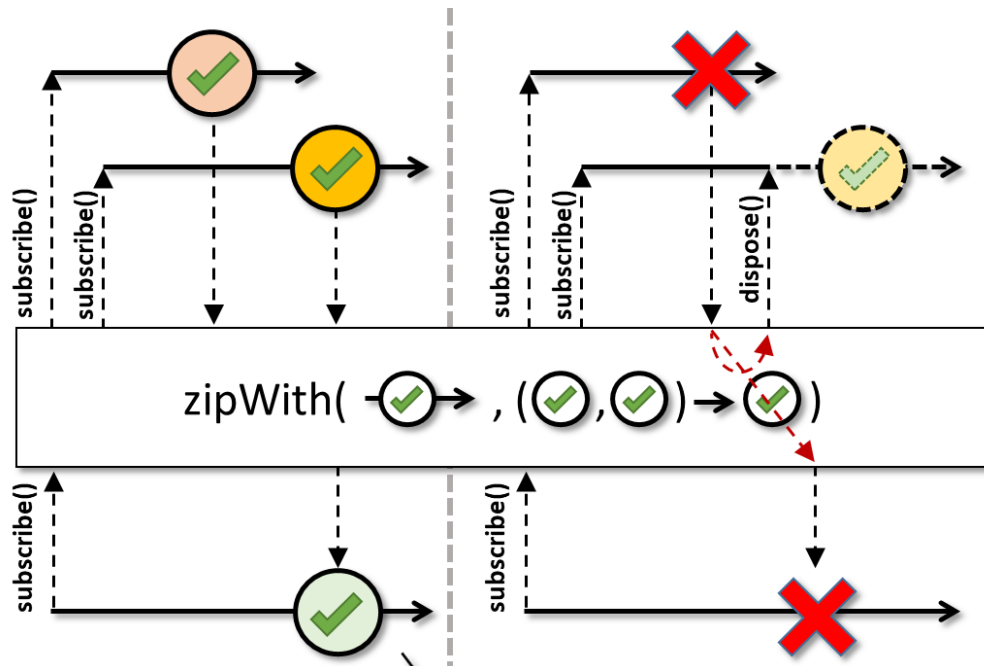
R - the result type

Functional Interface:

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

Applying Key Methods in the Single Class in ex3

- The zipWith() method
 - Joins two results into a single result after they both emit
 - Combine the result from this & other Single into another object via a given combinator bifunction

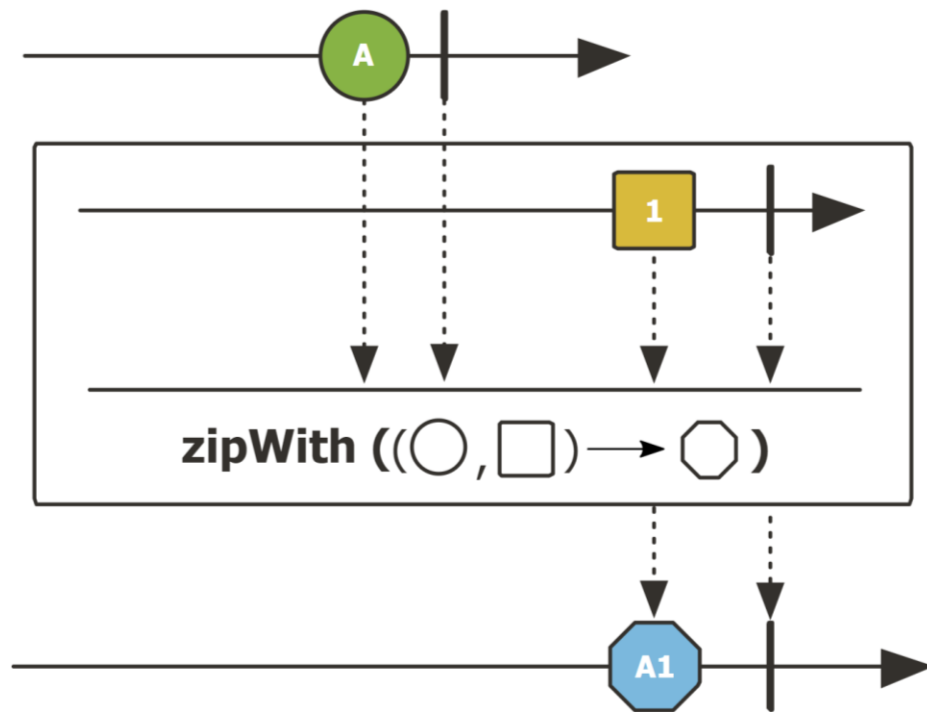


zipWith() can transform the type of elements it processes



Applying Key Methods in the Single Class in ex3

- The zipWith() method
 - Joins two results into a single result after they both emit
- Project Reactor's Mono.zipWith() works the same



Applying Key Methods in the Single Class in ex3

- The `zipWith()` method
 - Joins two results into a single result after they both emit
 - Project Reactor's `Mono.zipWith()` works the same
- Similar to the Java `CompletableFuture.thenCombine()` method

thenCombine

```
public <U,V> CompletableFuture<V> thenCombine(CompletionStage<? extends U> other,  
                                              BiFunction<? super T,? super U,? extends V> fn)
```

Description copied from interface: `CompletionStage`

Returns a new `CompletionStage` that, when this and the other given stage both complete normally, is executed with the two results as arguments to the supplied function. See the `CompletionStage` documentation for rules covering exceptional completion.

Specified by:

`thenCombine` in interface `CompletionStage<T>`

Type Parameters:

U - the type of the other `CompletionStage`'s result

V - the function's return type

Parameters:

other - the other `CompletionStage`

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

Returns:

the new `CompletionStage`

Applying Key Methods in the Single Class in ex3

- The `Schedulers.computation()` method
 - Hosts a fixed pool of single-threaded Executor Service-based workers that is suitable for parallel work

`static Scheduler computation()`

Applying Key Methods in the Single Class in ex3

- The `Schedulers.computation()` method
- Hosts a fixed pool of single-threaded Executor Service-based workers that is suitable for parallel work
- Optimized for fast running non-blocking operations
- i.e., computation-intensive *not* I/O-intensive!

Class Schedulers

```
java.lang.Object  
io.reactivex.rxjava3.schedulers.Schedulers
```

```
public final class Schedulers  
extends Object
```

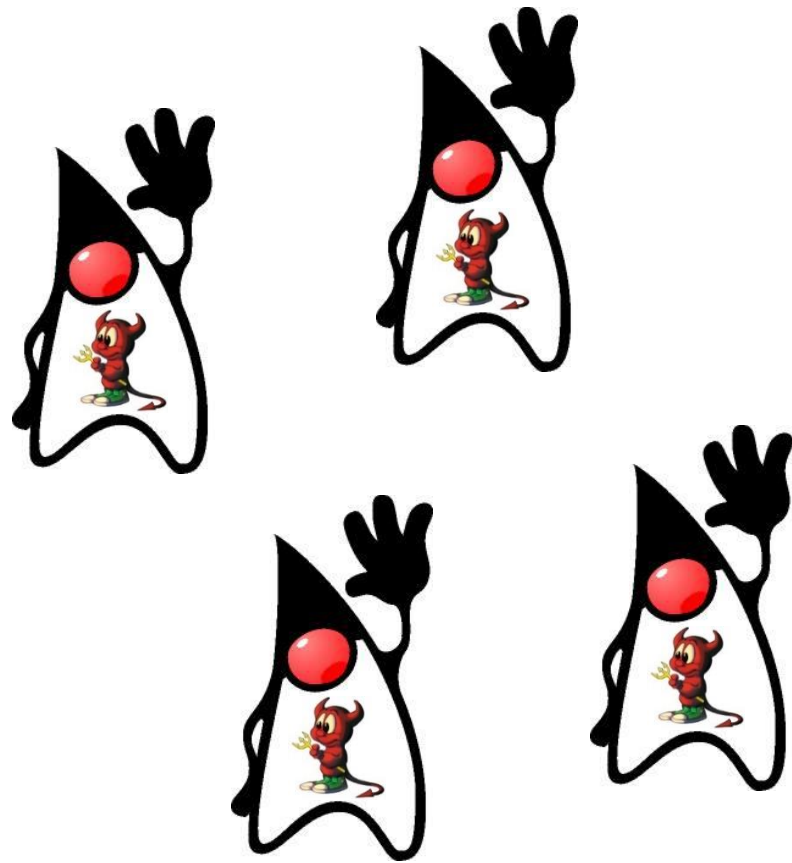
Static factory methods for returning standard Scheduler instances.

The initial and runtime values of the various scheduler types can be overridden via the `RxJavaPlugins.setInit(scheduler name)SchedulerHandler()` and `RxJavaPlugins.set(scheduler name)SchedulerHandler()` respectively.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html

Applying Key Methods in the Single Class in ex3

- The `Schedulers.computation()` method
 - Hosts a fixed pool of single-threaded Executor Service-based workers that is suitable for parallel work
 - Optimized for fast running non-blocking operations
 - Implemented via daemon threads that won't prevent the app from exiting even if its work isn't done



See www.baeldung.com/java-daemon-thread

Applying Key Methods in the Single Class in ex3

- The `Schedulers.computation()` method
 - Hosts a fixed pool of single-threaded `Executor Service`-based workers that is suitable for parallel work
- Project Reactor's `Schedulers.parallel()` method is similar

parallel

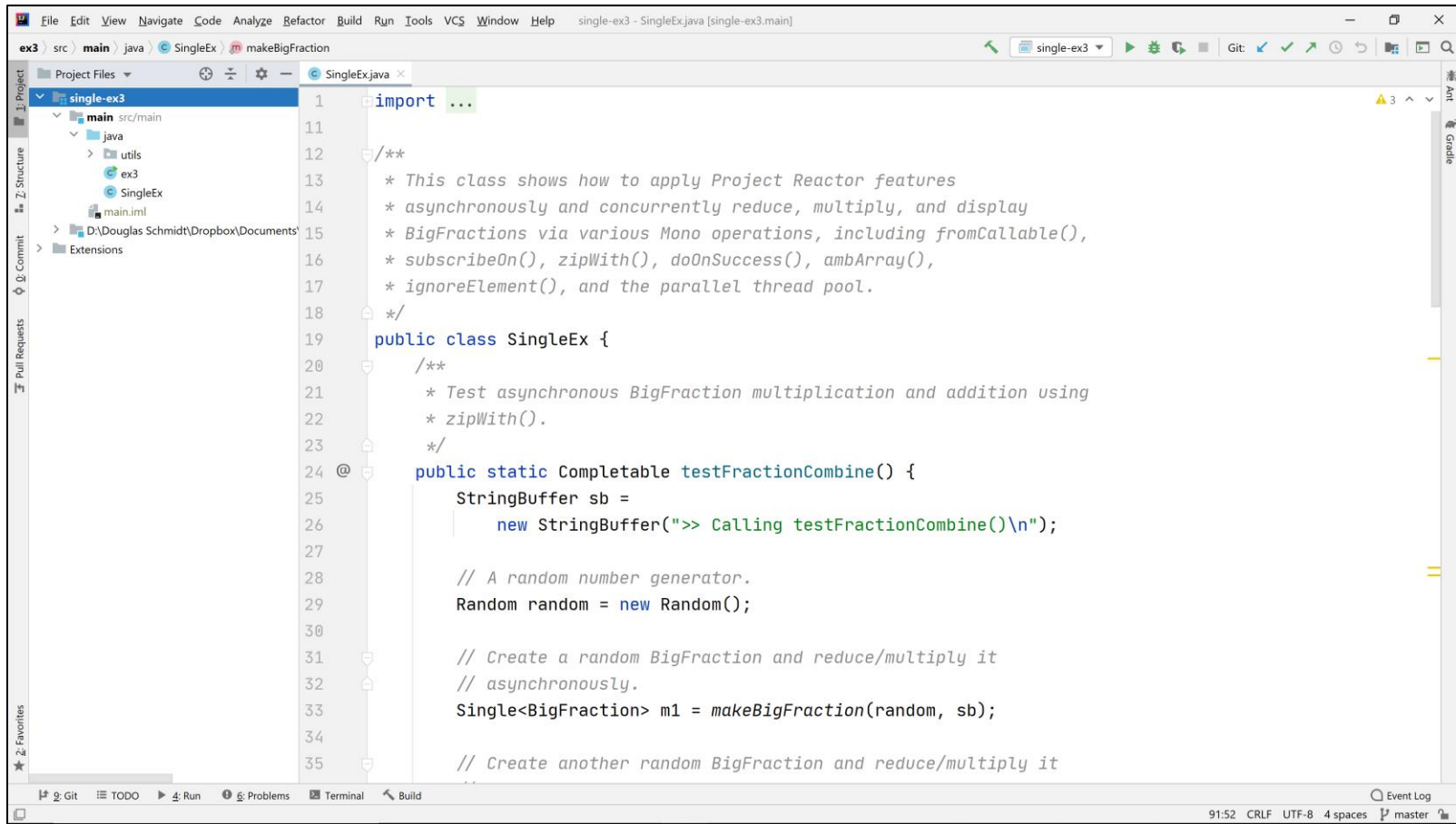
```
public static Scheduler parallel()
```

`Scheduler` that hosts a fixed pool of single-threaded `ExecutorService`-based workers and is suited for parallel work.

Returns:

default instance of a `Scheduler` that hosts a fixed pool of single-threaded `ExecutorService`-based workers and is suited for parallel work

Applying Key Methods in the Single Class in ex3



```
1  import ...
11
12  /**
13   * This class shows how to apply Project Reactor features
14   * asynchronously and concurrently reduce, multiply, and display
15   * BigFractions via various Mono operations, including fromCallable(),
16   * subscribeOn(), zipWith(), doOnSuccess(), ambArray(),
17   * ignoreElement(), and the parallel thread pool.
18   */
19  public class SingleEx {
20      /**
21       * Test asynchronous BigFraction multiplication and addition using
22       * zipWith().
23       */
24      @Test
25      public static Completable testFractionCombine() {
26          StringBuffer sb =
27              new StringBuffer(">> Calling testFractionCombine()\n");
28
29          // A random number generator.
30          Random random = new Random();
31
32          // Create a random BigFraction and reduce/multiply it
33          // asynchronously.
34          Single<BigFraction> m1 = makeBigFraction(random, sb);
35
36          // Create another random BigFraction and reduce/multiply it
```

See github.com/douglasraigschmidt/LiveLessons/tree/master/Reactive/Single/ex3

End of Applying Key Methods in the Single Class (Part 3)