Learning Objectives in this Part of the Lesson

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

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

```java
return Single
    .fromCallable(reduceFraction)
    .map(convertToMixedString)
    .doOnSuccess(printResult)
    .ignoreElement();
```

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 \textit{synchronously} to perform basic Single operations
• e.g., fromCallable(), doOnSuccess(), ignoreElement(), & map()

```
return Single
    .fromCallable(reduceFraction)
    .map(convertToMixedString)
    .doOnSuccess(printResult)
    .ignoreElement();
```
Applying Key Methods in the Single Class to `ex1`

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

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

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

```java
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](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)
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
Applying Key Methods in the Single Class to ex1

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

```java
<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

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

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](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

```java
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](https://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)
```

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](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's `thenAccept()` method

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#thenAccept](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#thenAccept)
• The ignoreElement() method

• Returns a Completable that ignores the success value of this Single & signals onComplete() instead
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

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Completable.html](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
java.lang.Object
util.AsycnTester

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**

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>register</td>
</tr>
<tr>
<td>(io.reactivex.rxjava3.core.Single&lt;java.lang.Long&gt;)</td>
<td>runTests()</td>
</tr>
</tbody>
</table>

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](http://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 unreduce big fraction.
        BigFraction unreducedFraction = BigFraction
            .valueOf(new BigInteger(sBI1),
                    new BigInteger(sBI2),
                    reduce: false);
    }
}
```

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