Applying Key Operators in the Mono Class: Case Study ex3

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 operators in the Mono class & know how they are applied in a detailed case study
  - Case study ex3 examines the fromCallable(), zipWith(), zip(), flatMap(), doOnSuccess(), subscribeOn(), then(), & parallel thread pool operators

```
Mono<BigFraction> m1 = makeBigFractionAsync(...)
Mono<BigFraction> m2 = makeBigFractionAsync(...);
Mono<BigFraction> m3 = makeBigFractionAsync(...);
Mono<BigFraction>[] asyncMult = new Mono[] {
  m1.flatMap(bf1 -> multiplyAsync(bf1, sBigReducedFraction)),
  m2.flatMap(bf2 -> multiplyAsync(bf2, sBigReducedFraction)),
  m3.flatMap(bf3 -> multiplyAsync(bf3, sBigReducedFraction))
};
...
return Mono.zip(combiner, asyncMult);
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/mono/ex3

Applying Key Operators in the Mono Class in ex3

Applying Key Operators in the Mono Class in ex3

```java
import static utils.BigFractionUtils.*;

/**
 * This class shows how to apply Project Reactor features
 * asynchronously to perform Mono operations, including
 * fromCallable(), subscribeOn(), zipWith(), doOnSuccess(), then(),
 * and the parallel thread pool.
 */
public class MonoEx {
    /**
     * Test asynchronous BigFraction multiplication and addition using
     * zipWith().
     */
    public static Mono<Void> testFractionCombine() {
        StringBuilder sb =
            new StringBuilder(">> Calling testFractionCombine()\n");

        // A random number generator.
        Random random = new Random();

        // Create a random BigFraction and reduce/multiply it
        // asynchronously.
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/mono/ex3

End of Applying Key Methods in the Mono Class: Case Study ex3