

Applying Key Operators in the Flowable Class: Case Study ex1

Douglas C. Schmidt

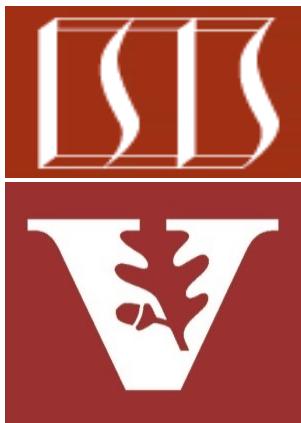
d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA



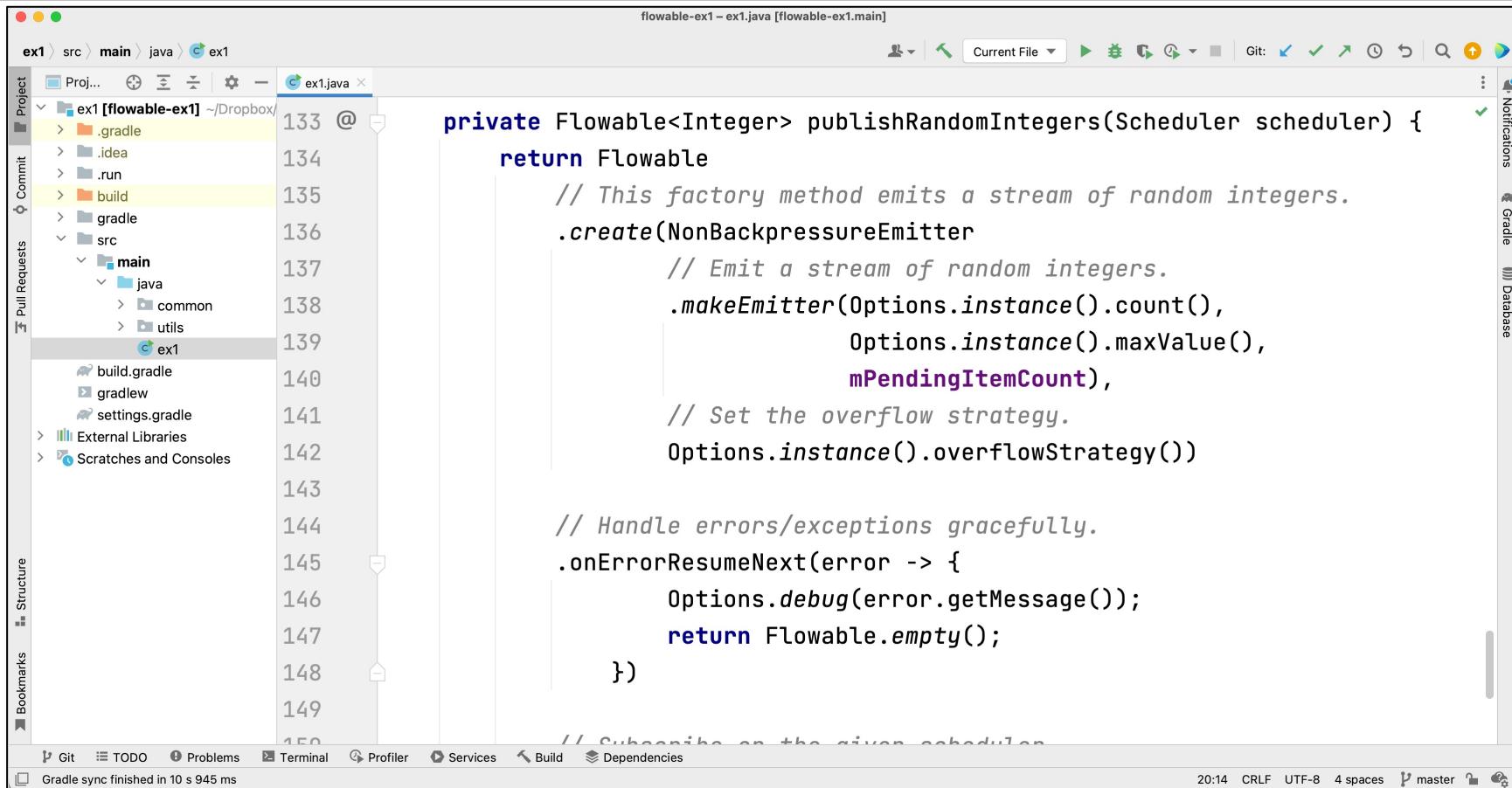
Learning Objectives in this Part of the Lesson

- Case study ex1 applies RxJava Flowable features to demonstrate various types of backpressure strategies (e.g., MISSING, BUFFER, ERROR, LATEST, & DROP) between a Publisher & a Subscriber that run in the context of different Scheduler objects

```
return Flowable
    .create(NonBackpressureEmitter
        .makeEmitter(count,
                      maxValue,
                      mPendingItemCount),
        overflowStrategy)
    .onErrorResumeNext(error -> {
        debug(error.getMessage());
        return Flowable.empty();
    })
    .subscribeOn(scheduler);
```

Applying Key Operators in the Flowable Class to ex1

Applying Key Operators in the Flowable Class to ex1



The screenshot shows a Java code editor in an IDE. The project structure on the left shows a file named `ex1.java` under the `src/main/java/ex1` path. The code itself is a factory method for a `Flowable<Integer>` object. It uses the `Flowable.create` method with a `NonBackpressureEmitter`. The emitter is configured to emit random integers with a count of 100, a maximum value of 1000, and a pending item count of 10. It also sets an overflow strategy to `Options.instance().overflowStrategy()`. The code handles errors by logging the message and returning an empty `Flowable`. The code is annotated with comments explaining its purpose. The code editor has a status bar at the bottom showing build and sync information.

```
private Flowable<Integer> publishRandomIntegers(Scheduler scheduler) {
    return Flowable
        // This factory method emits a stream of random integers.
        .create(NonBackpressureEmitter
            // Emit a stream of random integers.
            .makeEmitter(Options.instance().count(),
                Options.instance().maxValue(),
                mPendingItemCount),
            // Set the overflow strategy.
            Options.instance().overflowStrategy())
        // Handle errors/exceptions gracefully.
        .onErrorResumeNext(error -> {
            Options.debug(error.getMessage());
            return Flowable.empty();
        })
        // Subscribes on the given scheduler
}
```

End of Applying Key Operators in the Flowable Class: Case Study ex1