Reactive Streams

Introduced in Java 9

Java 9 Flow

https://docs.oracle.com/javase/9/docs/api/java/util/concurrent/Flow.html http://www.reactive-streams.org/

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP javabrains.io

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES SEARCH: Search

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module java.base
Package java.util.concurrent

Class Flow

java.lang.Object
  java.util.concurrent.Flow

```
public final class Flow
extends Object
```

Interrelated interfaces and static methods for establishing flow-controlled components in which `Publishers` produce items consumed by one or more `Subscribers`, each managed by a `Subscription`.

These interfaces correspond to the reactive-streams specification. They apply in both concurrent and distributed asynchronous settings: All (seven) methods are defined in `void` "one-way" message style. Communication relies on a simple form of flow control (method `Flow.Subscription.request(long)`) that can be used to avoid resource management problems that may otherwise occur in "push" based systems.

Examples. A `Flow.Publisher` usually defines its own `Flow.Subscription` implementation; constructing one in method `subscribe` and issuing it to the calling `Flow.Subscriber`. It publishes items to the subscriber asynchronously, normally using an `Executor`. For example, here is a very simple publisher that only issues (when requested) a single TRUE item to a single subscriber. Because the subscriber receives only a single item, this class does not use buffering and ordering control required in most implementations (for example `SubmissionPublisher`).

```
class OneShotPublisher implements Publisher<Boolean> {
  private final ExecutorService executor = ForkJoinPool.commonPool(); // daemon-based
```

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP javabrains.io

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES SEARCH: Search

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Nested Classes

Modifier and Type	Class	Description
static interface	`Flow.Processor<T,R>`	A component that acts as both a Subscriber and Publisher.
static interface	`Flow.Publisher<T>`	A producer of items (and related control messages) received by Subscribers.
static interface	`Flow.Subscriber<T>`	A receiver of messages.
static interface	`Flow.Subscription`	Message control linking a `Flow.Publisher` and `Flow.Subscriber`.

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method	Description
static int	`defaultBufferSize()`	Returns a default value for Publisher or Subscriber buffering, that may be used in the absence of other constraints.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Three abstractions

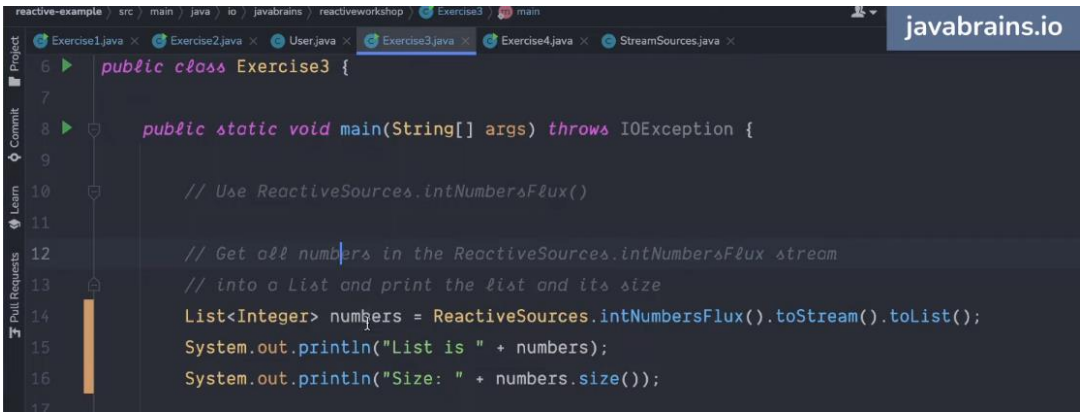
- Publisher
- Subscriber
- Subscription

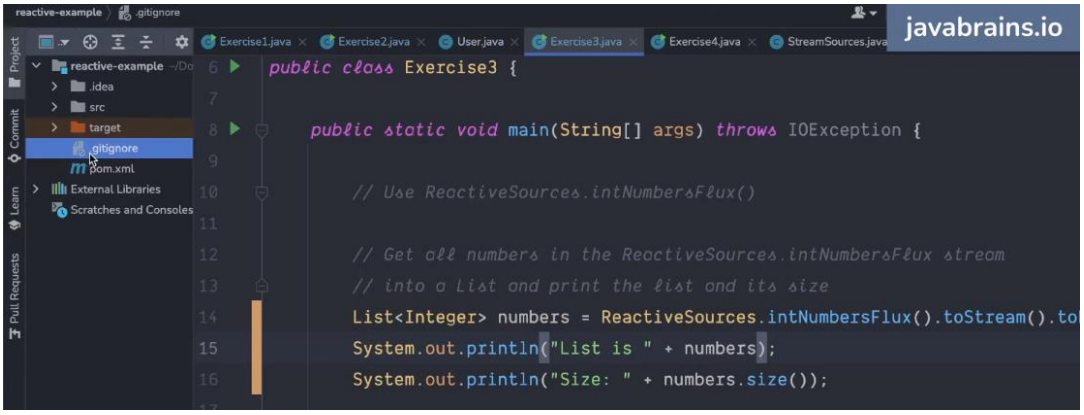
Interfaces only!

You don't use any of these directly

What you will use

- Flux
- Mono





```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.ap
    <modelVersion>4.0.0</modelVersion>

    <groupId>io.javabrains</groupId>
    <artifactId>reactive-example</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>io.projectreactor</groupId>
            <artifactId>reactor-core</artifactId>
            <version>3.4.14</version>
        </dependency>
    </dependencies>
```

```
    <artifactId>reactive-example</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>io.projectreactor</groupId>
            <artifactId>reactor-core</artifactId>
            <version>3.4.14</version>
        </dependency>
    </dependencies>

</project>
```

We are using the Reactor Core library to get access to the Flux and Mono class implementations to construct the Flux and Mono stream types

javabrains.io

```java
package io.javabrains.reactiveworkshop;

import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import java.time.Duration;

/**
 * This class is a source of reactive streams used in the exercises.
 * DO NOT MODIFY THIS CODE
 *
 * @author koushikkothagal
 */
public class ReactiveSources {

    public static Flux<String> stringNumbersFlux() {
        return Flux.just("one", "two", "three", "four", "five", "six", "seven"
                .delayElements(Duration.ofSeconds(1));
    }

    public static Flux<Integer> intNumbersFlux() {
        return Flux
                .range(1, 10)
```

javabrains.io

```java
public class Exercise3 {

    public static void main(String[] args) throws IOException {

        // Use ReactiveSources.intNumbersFlux()

        // Get all numbers in the ReactiveSources.intNumbersFlux stream
        // into a List and print the list and its size
        List<Integer> numbers = ReactiveSources.intNumbersFlux().toStream().to
        System.out.println("List is " + numbers);
        System.out.println("Size: " + numbers.size());
```