Overview of the Java Reactive Streams API

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

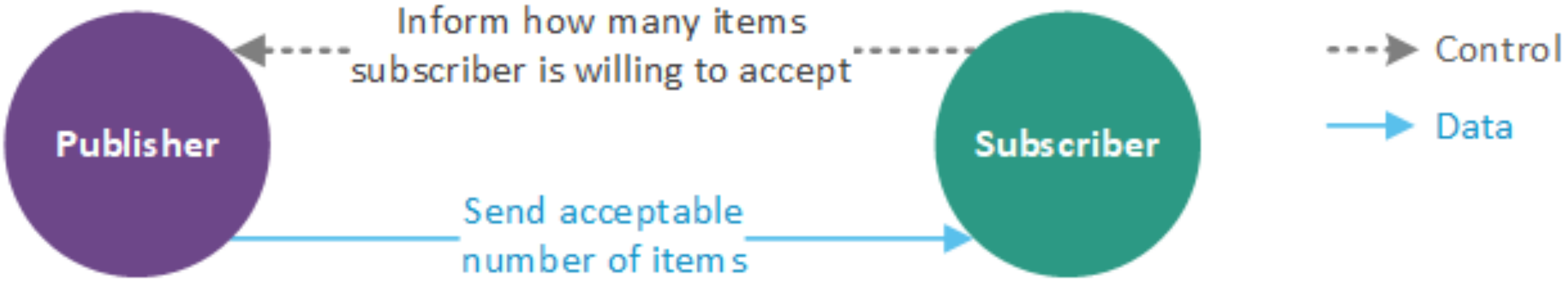
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

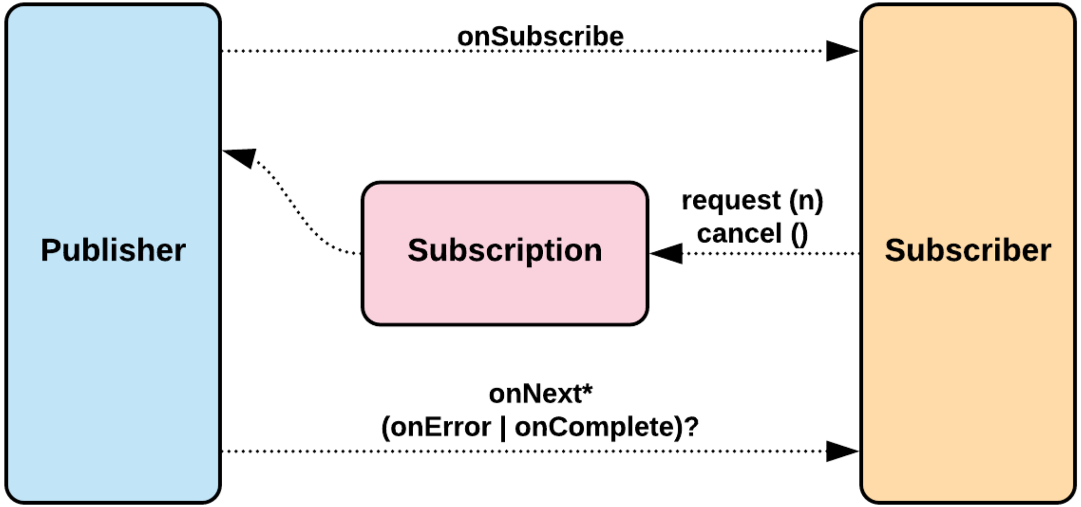
- Understand the key benefits & principles underlying the reactive programming paradigm
- Know the Java reactive streams API


Inform how many items subscriber is willing to accept
Publisher
Subscriber
Send acceptable number of items
Control
Data

See community.oracle.com/docs/DOC-1006738

Learning Objectives in this Part of the Lesson

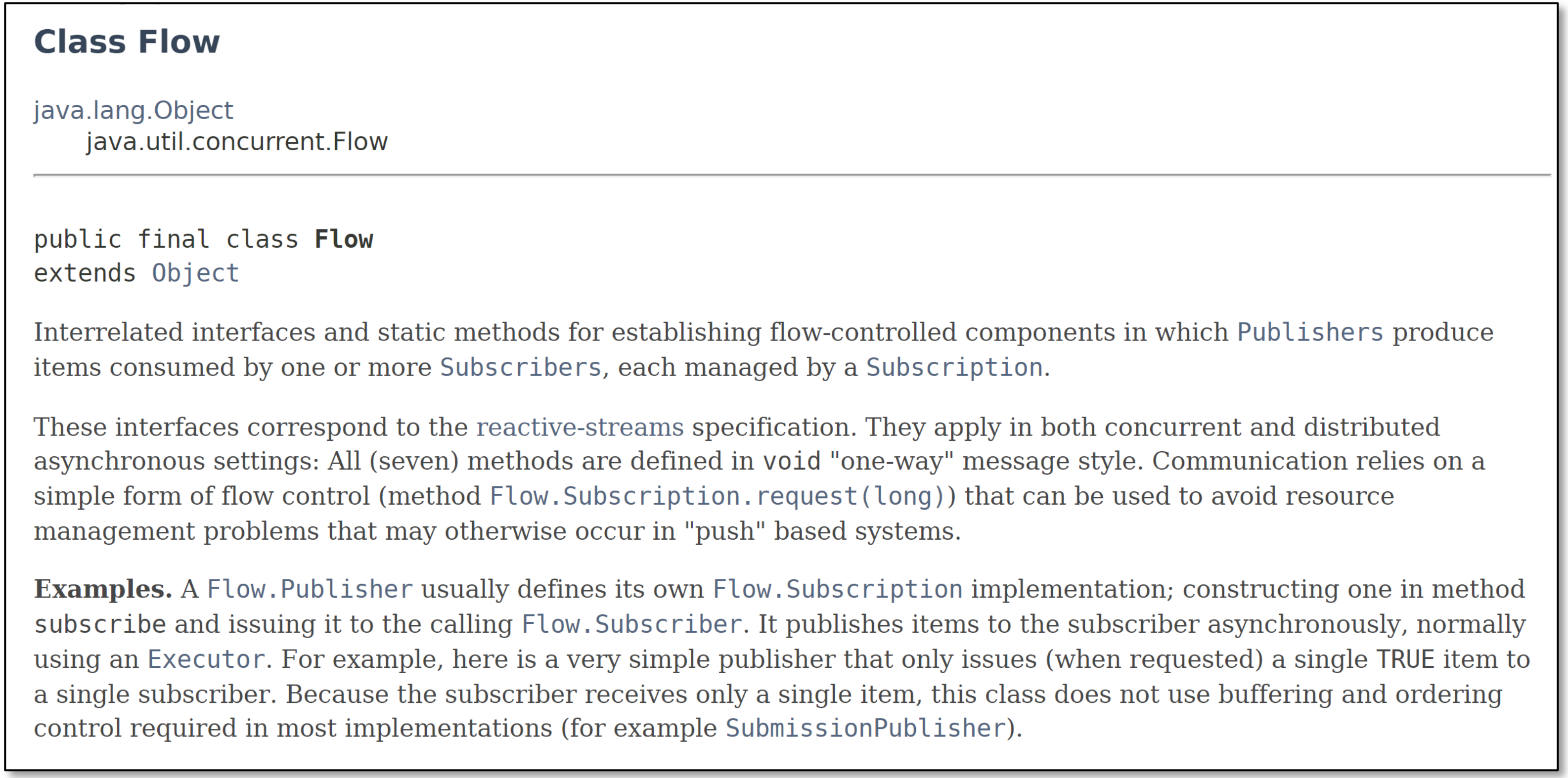
- Understand the key benefits & principles underlying the reactive programming paradigm
- Know the Java reactive streams API
  - Recognize key abstractions



Reactive Programming & Java Reactive Streams

Reactive Programming & Java Reactive Streams

- Java 9+ supports reactive programming via Reactive Streams & the Flow API



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`.

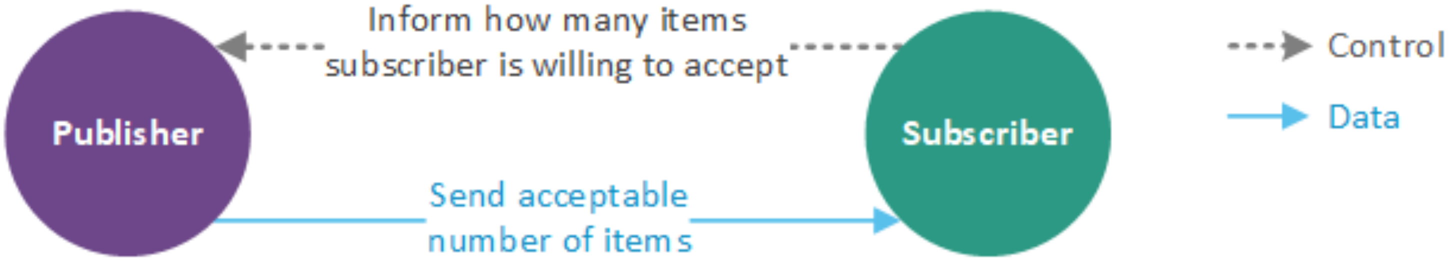
These interfaces correspond to the reactive-streams specification. They apply in both concurrent and distributed asynchronous settings: All (seven) methods are defined in `void` "one-way" message style. Communication relies on a simple form of flow control (method `Flow.Subscription.request(long)`) that 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`).

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/Flow.html

Reactive Programming & Java Reactive Streams

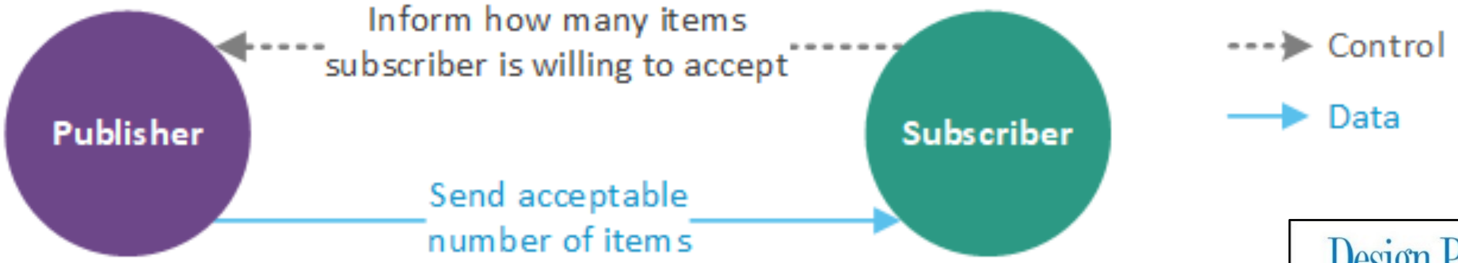
- Java 9+ supports reactive programming via Reactive Streams & the Flow API
  - Adds support for stream-oriented pub/sub patterns


Inform how many items subscriber is willing to accept
Publisher
Subscriber
Send acceptable number of items
Control
Data

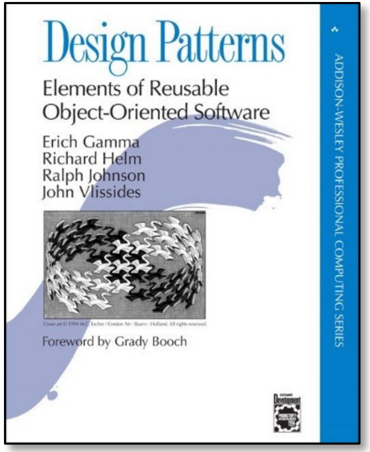
See javasampleapproach.com/java/java-9/java-9-flow-api-example-publisher-and-subscriber

Reactive Programming & Java Reactive Streams

- Java 9+ supports reactive programming via Reactive Streams & the Flow API
  - Adds support for stream-oriented pub/sub patterns


Inform how many items subscriber is willing to accept
Publisher
Subscriber
Send acceptable number of items
Control
Data

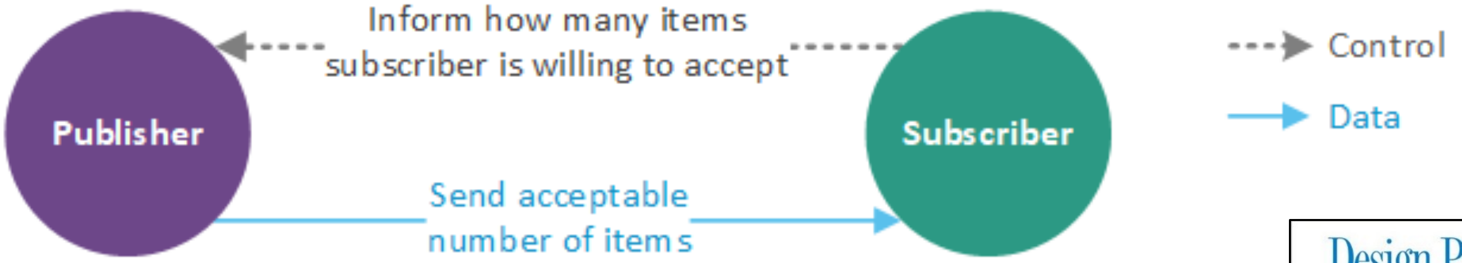
- Combines two patterns


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

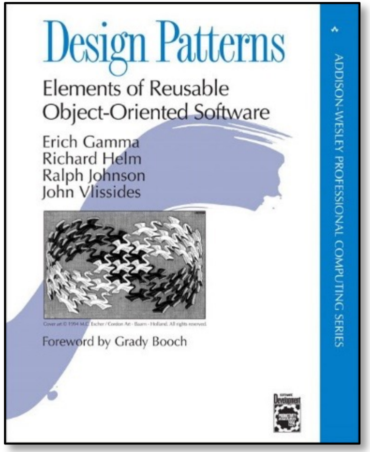
See www.journaldev.com/20723/java-9-reactive-streams

Reactive Programming & Java Reactive Streams

- Java 9+ supports reactive programming via Reactive Streams & the Flow API
  - Adds support for stream-oriented pub/sub patterns


Inform how many items subscriber is willing to accept
Publisher
Subscriber
Send acceptable number of items
Control
Data

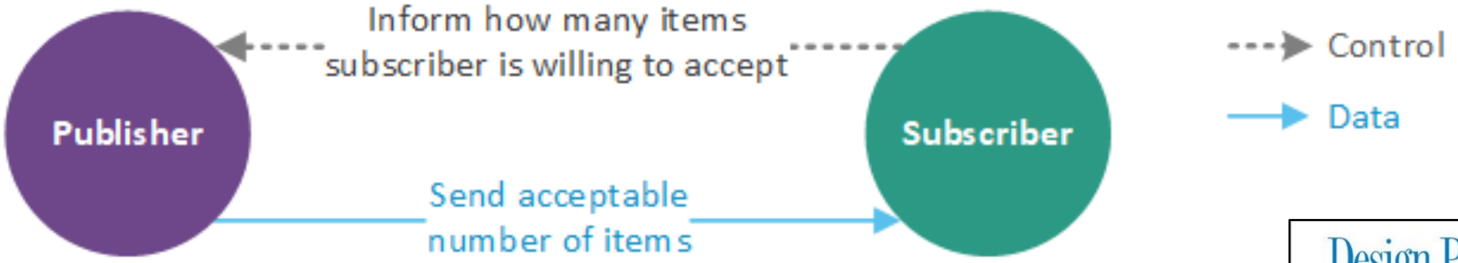
- Combines two patterns
  - *Iterator*, which applies a "pull model" where app subscriber(s) pull items from a publisher source


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

See en.wikipedia.org/wiki/Iterator_pattern

Reactive Programming & Java Reactive Streams

- Java 9+ supports reactive programming via Reactive Streams & the Flow API
  - Adds support for stream-oriented pub/sub patterns



Inform how many items subscriber is willing to accept

Publisher

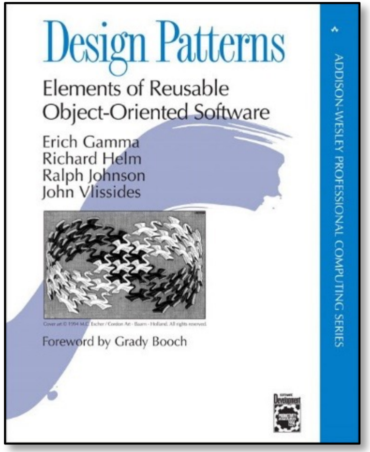
Subscriber

Send acceptable number of items

Control

Data

- Combines two patterns
  - *Iterator*, which applies a "pull model" where app subscriber(s) pull items from a publisher source
  - *Observer*, which applies a "push model" that reacts when a publisher source pushes an item to subscriber sink(s)



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

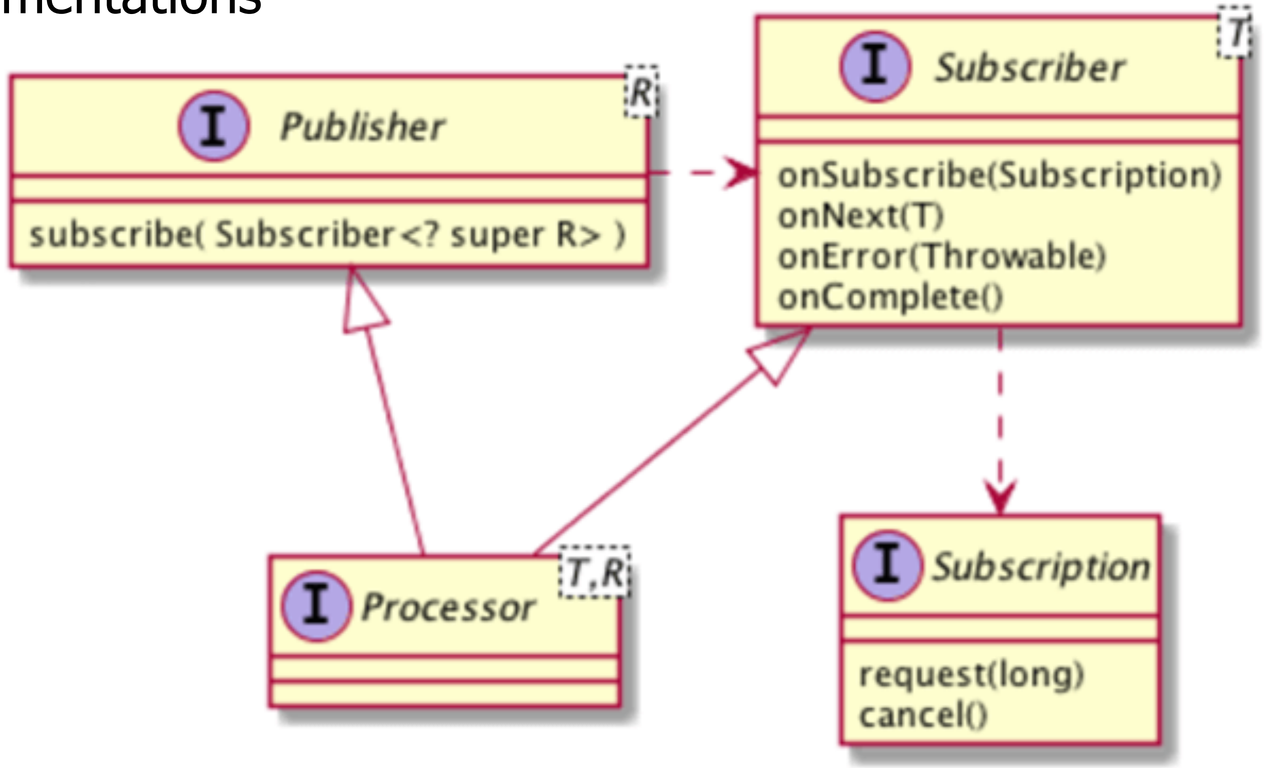
Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

See en.wikipedia.org/wiki/Observer_pattern

Reactive Programming & Java Reactive Streams

- The Java Flow API defines interfaces designed to ensure interoperability of reactive streams implementations

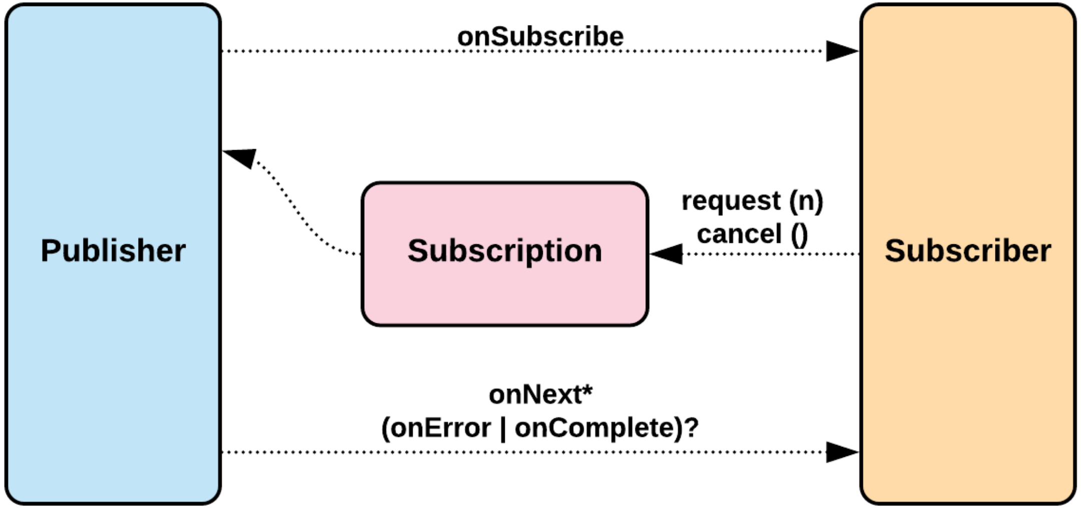


See www.reactive-streams.org

Key Abstractions in the Java Flow API

Key Abstractions in the Java Flow API

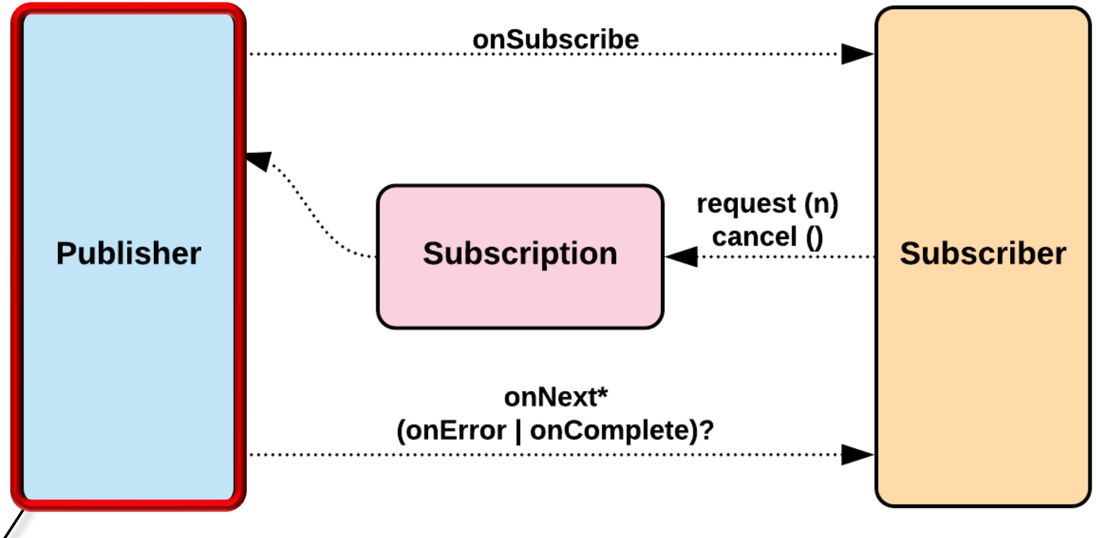
- A "flow" involves interactions between three key abstractions



See www.baeldung.com/java-9-reactive-streams

Key Abstractions in the Java Flow API

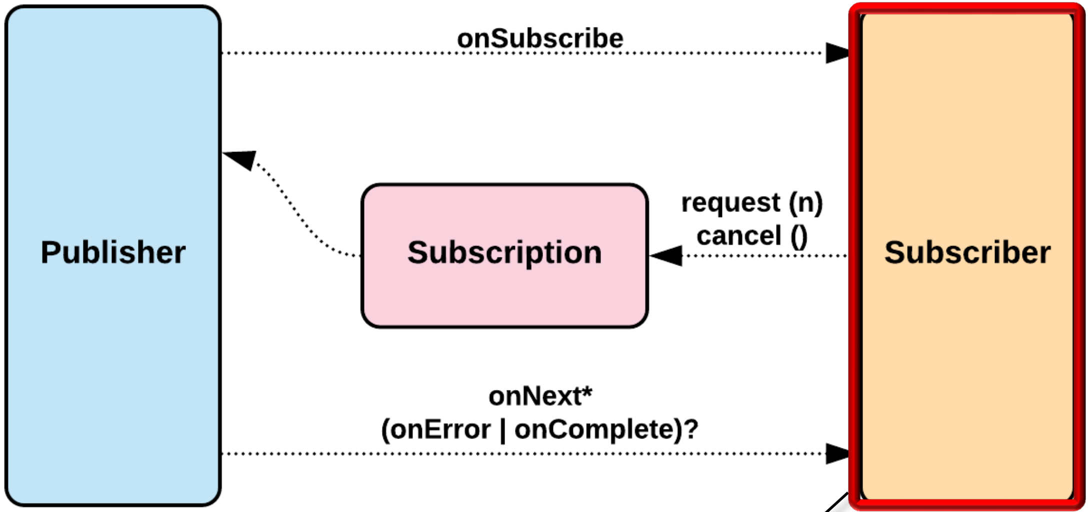
- A “flow” involves interactions between three key abstractions



1. Publisher(s) are sources that produce 0+ events that can be pushed to subscriber(s)

Key Abstractions in the Java Flow API

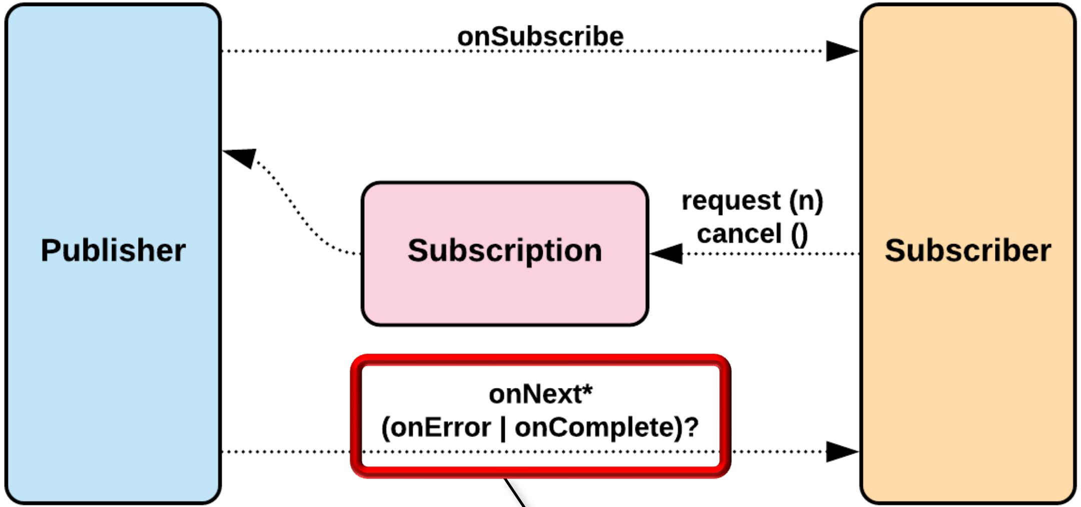
- A "flow" involves interactions between three key abstractions



2. Subscriber(s) are sinks that register for & consume events pushed by publisher(s)

Key Abstractions in the Java Flow API

- A “flow” involves interactions between three key abstractions

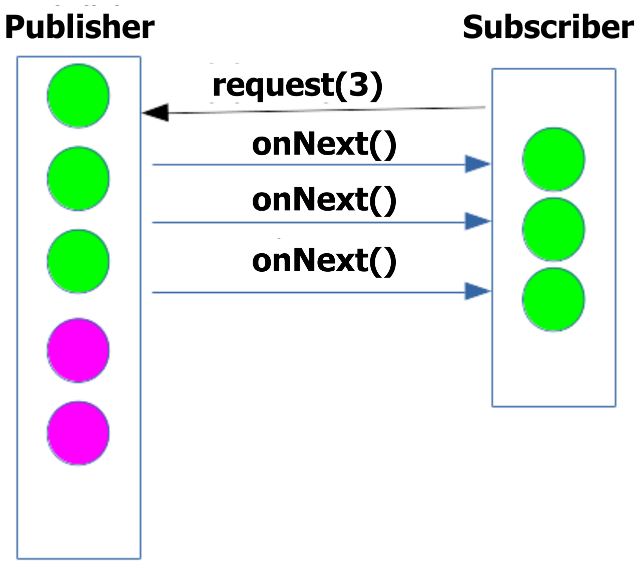


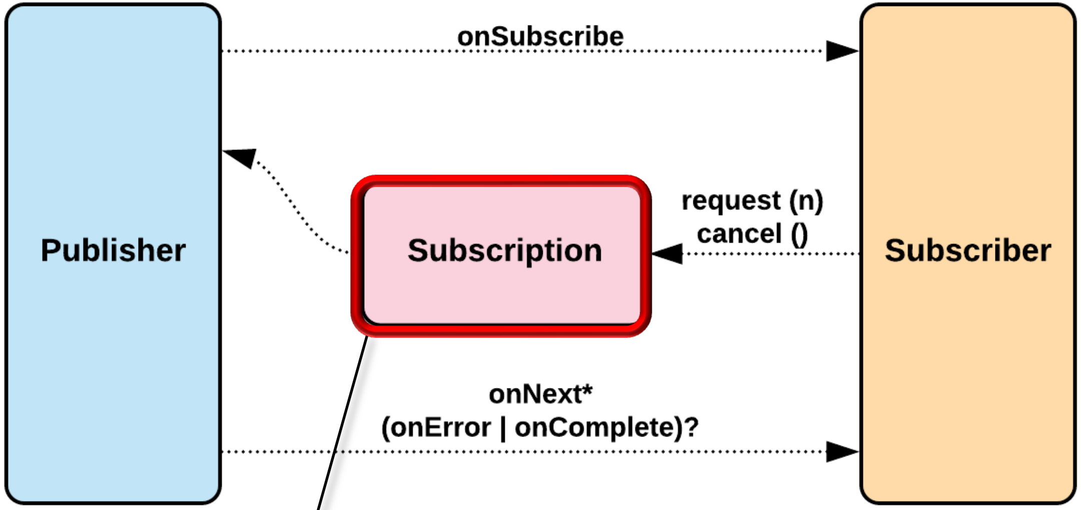
Publisher(s) push events to registered subscriber(s) by invoking hook methods

See wiki.c2.com/?HookMethod

Key Abstractions in the Java Flow API

- A “flow” involves interactions between three key abstractions



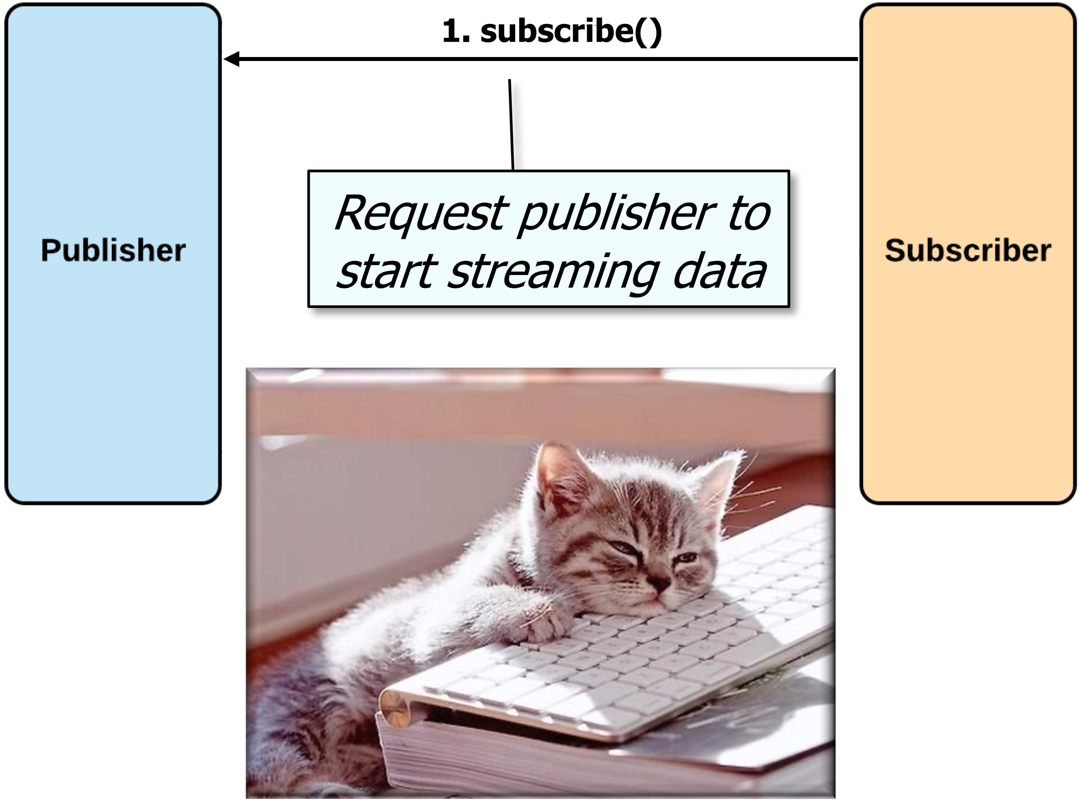


3. Subscription is used to control the flow of events between a subscriber & a publisher

See en.wikipedia.org/wiki/Flow_control_(data)

Key Abstractions in the Java Flow API

- A “flow” involves interactions between three key abstractions

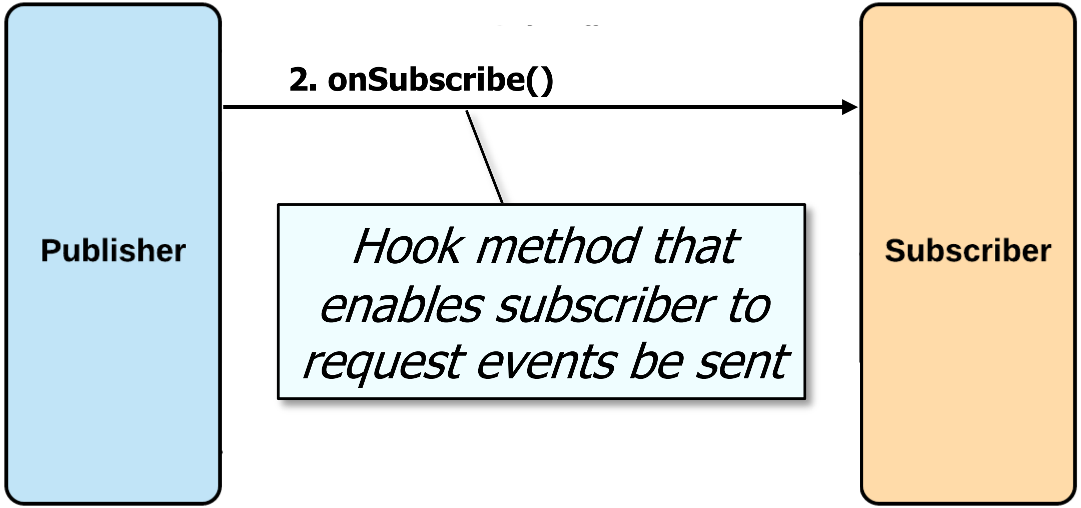


Request publisher to start streaming data

A reactive stream is “lazy” & just starts processing when subscribe() is called

Key Abstractions in the Java Flow API

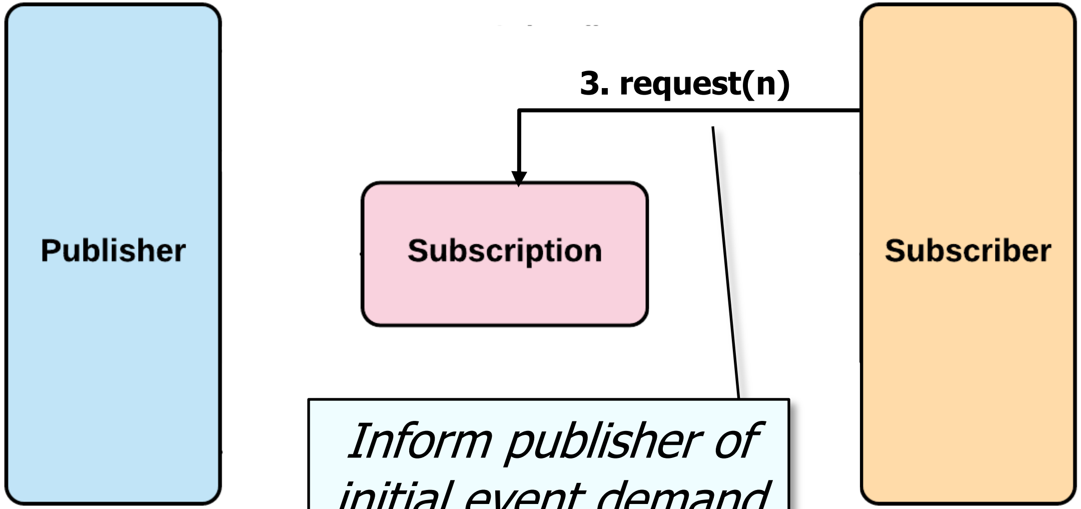
- A "flow" involves interactions between three key abstractions



Hook method that enables subscriber to request events be sent

Key Abstractions in the Java Flow API

- A “flow” involves interactions between three key abstractions

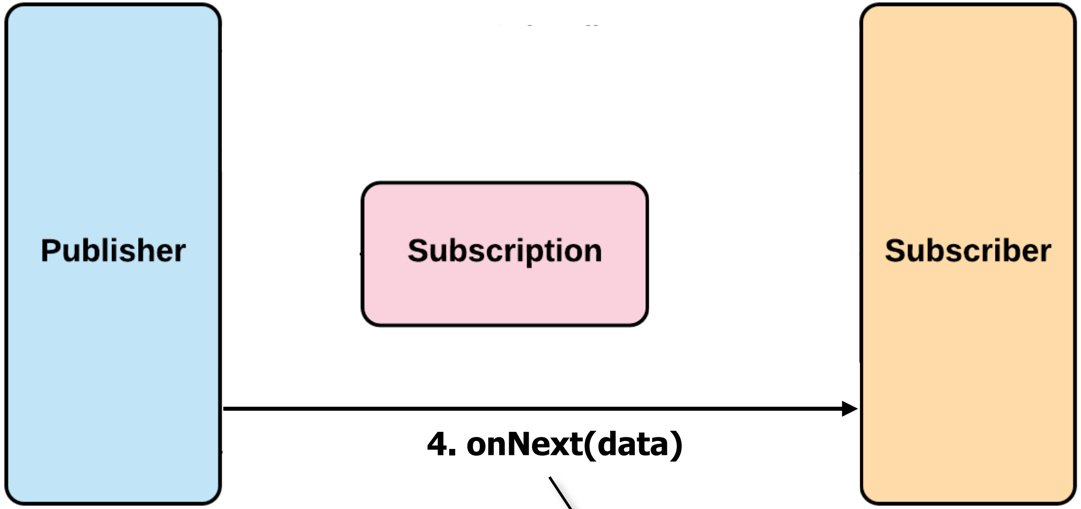


Inform publisher of initial event demand

No events are sent by a publisher until demand is signaled via this method

Key Abstractions in the Java Flow API

- A “flow” involves interactions between three key abstractions

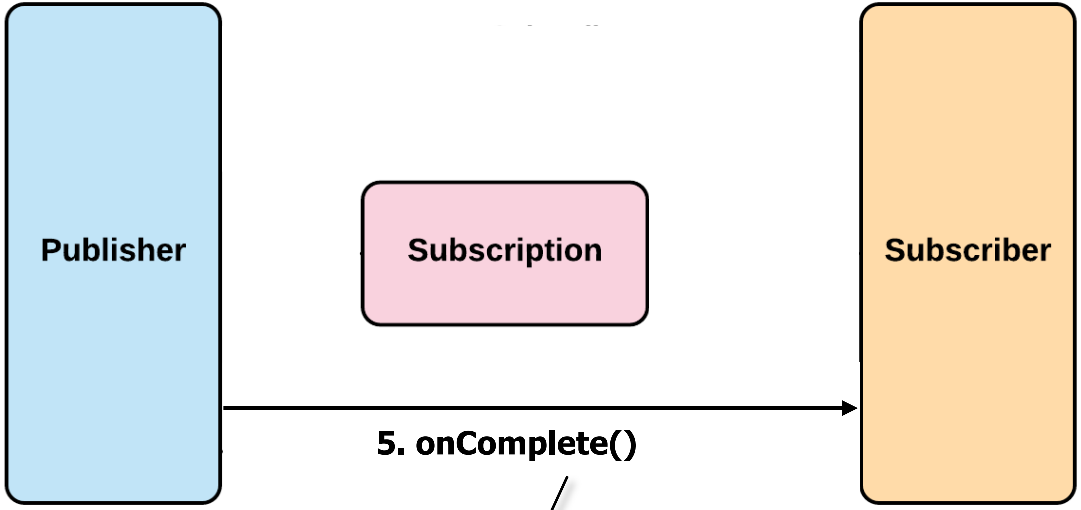


Data notification hook method called by the publisher in response to requests

There can be 0 or more onNext() notifications, which form a “stream”

Key Abstractions in the Java Flow API

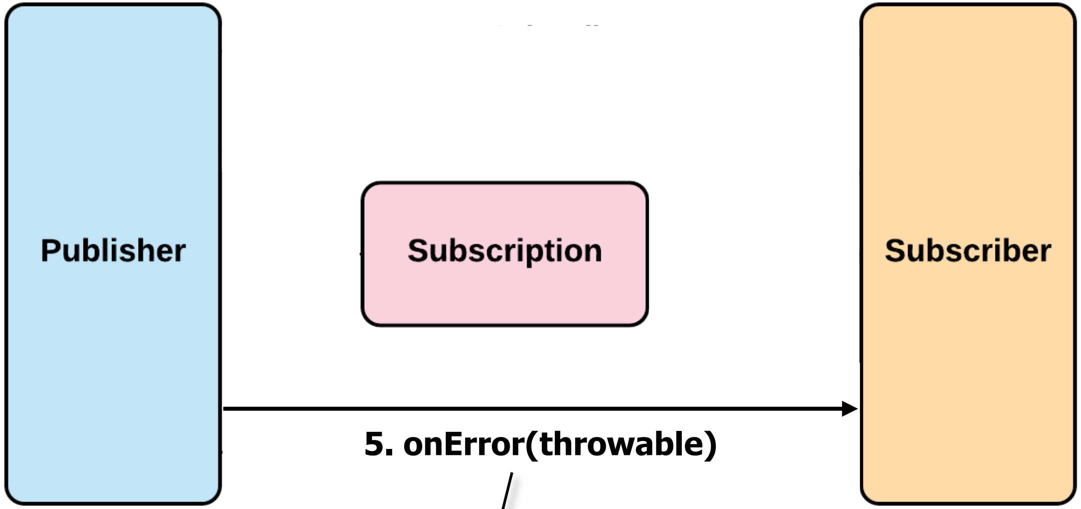
- A "flow" involves interactions between three key abstractions



Hook method called by publisher when all events have been sent successfully

Key Abstractions in the Java Flow API

- A "flow" involves interactions between three key abstractions



Hook method called by a publisher when an error occurs to convey the exception

End of Overview of the Java Reactive Streams API