The Structure & Functionality of the RSocket Quotes Backpressure Client

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

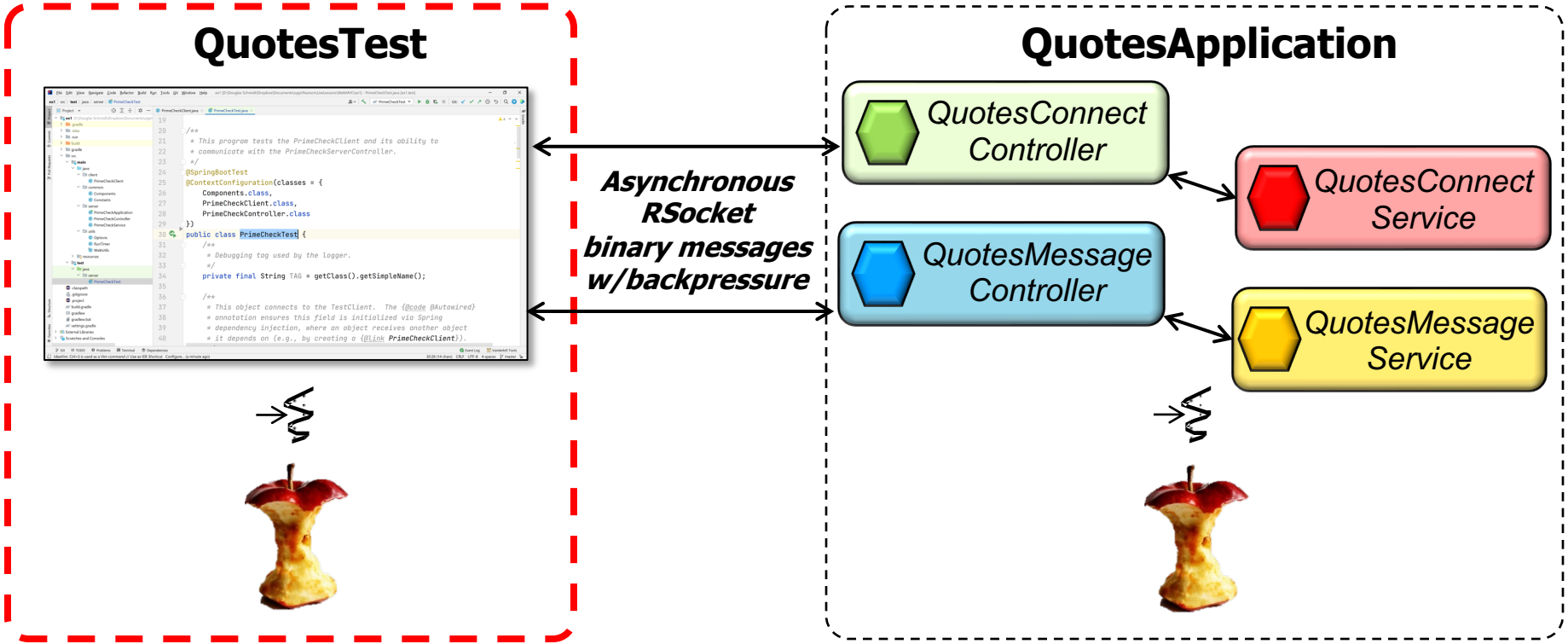
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Lesson

- Understand the structure & functionality of the RSocket Quotes backpressure client that asynchronously connects to & exchanges messages with the server

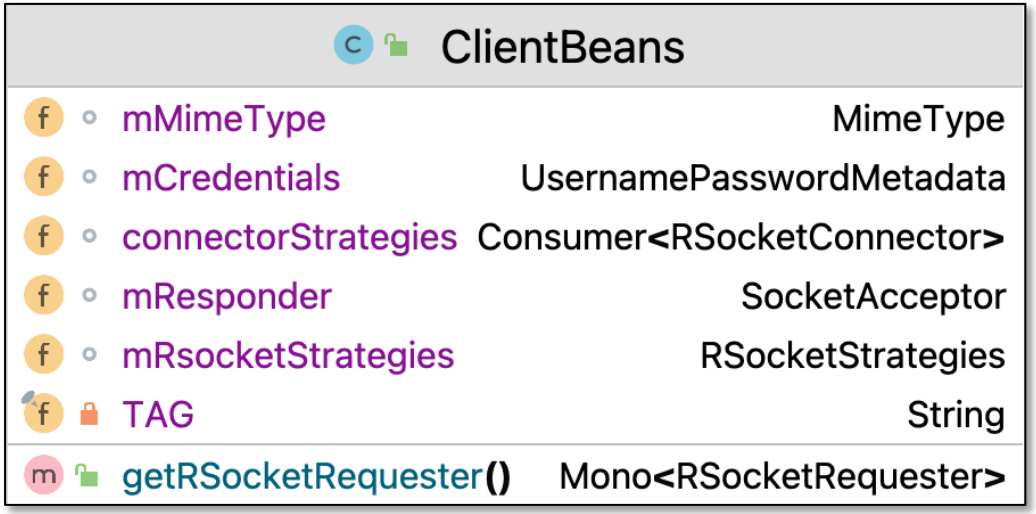


See github.com/douglascraigschmidt/LiveLessons/tree/master/RSocket/ex2

Overview of the Quotes Client Connection Initiator

Overview of the Quotes Client Connection Initiator

- The ClientBeans class defines a @Bean that returns a Mono emitting an RSocketRequester connected to the server



See RSocket/ex2/src/test/java/quotes/client/ClientBeans.java

Overview of the Quotes Client Connection Initiator

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
                getRequester(...) {
   return RSocketRequester
      .builder()
      .dataMimeType(...)
      .rsocketStrategies(...)
      .rsocketConnector(...)
      .setupRoute(...)
      .setupData(...)
      .setupMetadata(...)
      .tcp(...);
```

See en.wikipedia.org/wiki/Fluent_interface

Overview of the Quotes Client Connection Initiator

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls

```
Mono<RSocketRequester>
                getRequester(...) {
  return RSocketRequester
    .builder()
    ...
```

Obtain a builder to create a client Rsocket Requester by connecting to an RSocket server

See springframework/messaging/rsocket/RSocketRequester.html#builder

Overview of the Quotes Client Connection Initiator

- To obtain an RSocketRequester on the client side involves a fluent multi-step chain of calls
  - The mime type for data on the connection can be set

```
Mono<RSocketRequester>
                getRequester(...) {
  return RSocketRequester
    .builder()

    .dataMimeType(MediaType
                  .APPLICATION_CBOR)
    ...
```

Use the concise binary object representation (CBOR) protocol

See cbor.io

Overview of the ZippyQuotes Client Connection Initiator

- RSocketRequester.Builder accepts RSocketStrategies to configure the requester

```
Mono<RSocketRequester>
            getRequester(...) {
  return RSocketRequester
    .builder()
    ...
    .rsocketStrategies
      (rSsocketStrategies)
    ...
```

Select the Jackson CBOR encoder/decode as the protocol

```
var rSocketStrategies = RSocketStrategies
   .builder()
   .encoders(encoders -> encoders.add(new Jackson2CborEncoder()))
   .decoders(decoders -> decoders.add(new Jackson2CborDecoder()))
   .encoder(new SimpleAuthenticationEncoder())
   .build();
```

See springframework/http/codec/cbor/Jackson2CborEncoder.html

Overview of the ZippyQuotes Client Connection Initiator

- RSocketRequester.Builder accepts RSocketStrategies to configure the requester

```
Mono<RSocketRequester>
                getRequester(...) {
  return RSocketRequester
    .builder()
    ...
    .rsocketStrategies
      (rSsocketStrategies)
    ...
```

Sends client's authentication credentials to the server

```
var rSocketStrategies = RSocketStrategies
   .builder()
   .encoders(encoders -> encoders.add(new Jackson2CborEncoder()))
   .decoders(decoders -> decoders.add(new Jackson2CborDecoder()))
   .encoder(new SimpleAuthenticationEncoder())
   .build();
```

See springframework/security/rsocket/metadata/SimpleAuthenticationEncoder.html

Overview of the Quotes Client Connection Initiator

- RSocketRequester.Builder provides a callback that exposes the under lying RSocketConnector for further configuration options

```
Mono<RSocketRequester>
            getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .rsocketConnector(
      connectorStrategies::
      accept)
    ...
```

```
Consumer<RSocketConnector> connectorStrategies =
    connector -> connector
    .reconnect(Retry.fixedDelay(2, ofSeconds(2)))
    .acceptor(mResponder);
```

Overview of the Quotes Client Connection Initiator

- RSocketRequester.Builder provides a callback that exposes the under lying RSocketConnector for further configuration options, e.g.
  - Keepalive intervals, interceptors, re-connection policy, acceptors, etc.

```
Mono<RSocketRequester>
            getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .rsocketConnector(
       connectorStrategies::
       accept)
    ...
```

Try to reconnect twice, with a delay of 2 seconds per retry

```
Consumer<RSocketConnector> connectorStrategies =
    connector -> connector
    .reconnect(Retry.fixedDelay(2, ofSeconds(2)))
    .acceptor(mResponder);
```

Overview of the Quotes Client Connection Initiator

- RSocketRequester.Builder provides a callback that exposes the under lying RSocketConnector for further configuration options, e.g.
  - Keepalive intervals, interceptors, re-connection policy, acceptors, etc.

```
SocketAcceptor mResponder =
   RSocketMessageHandler
      .responder(mRsocketStrategies,
                 new
          ConnectResponseHandler());
```

Receives server's response to the connection request

```
Consumer<RSocketConnector> connectorStrategies =
     connector -> connector
     .reconnect(Retry.fixedDelay(2, ofSeconds(2)))
     .acceptor(mResponder);
```

See javadoc.io/static/io.rsocket/rsocket-core/1.1.1/io/rsocket/SocketAcceptor.html

Overview of the Quotes Client Connection Initiator

- There are also methods the client uses to connect with the server & authenticate itself

```
Mono<RSocketRequester>
                getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .setupRoute(SERVER_CONNECT)
    ...
```

Set up the route to connect with the server

See earlier discussion about @ConnectMapping

Overview of the Quotes Client Connection Initiator

- There are also methods the client uses to connect with the server & authenticate itself

```
Mono<RSocketRequester>
            getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .setupMetadata
       (mCreds,
        mMimeType)
    ...
```

```
UsernamePasswordMetadata mCreds
  = new UsernamePasswordMetadata
    ("d.schmidt@vanderbilt.edu",
     "you-shall-not-pass");

MimeType mMimeType = MimeTypeUtils
  .parseMimeType(MESSAGE_RSOCKET_AUTHENTICATION.getString());
```

Set up the metadata to pass the credentials

Overview of the Quotes Client Connection Initiator

- There are also methods the client uses to connect with the server & authenticate itself

```
Mono<RSocketRequester>
                getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .setupData(UUID
                    .randomUUID()
                    .toString())
    ...
```

Set up the data payload to send the server initially

Overview of the Quotes Client Connection Initiator

- Finally, the chain must connect with the server to obtain a Mono to an RSocketRequester

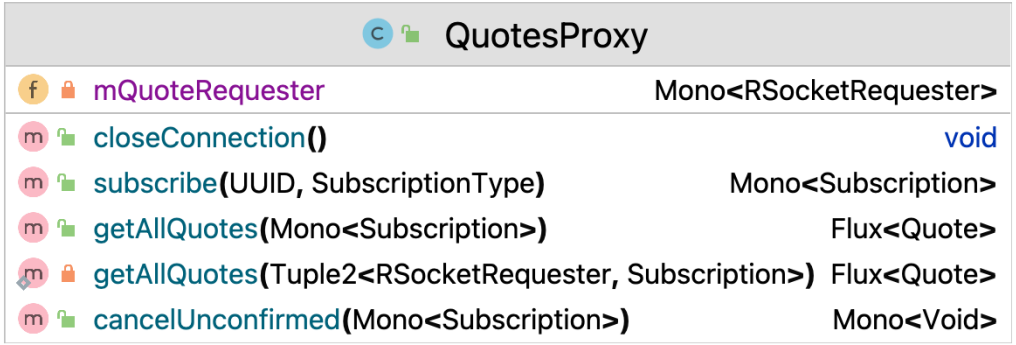
```
Mono<RSocketRequester>
                getRequester(...) {
  ...
  return RSocketRequester
    .builder()
    ...
    .tcp(LOCAL_HOST,
         SERVER_PORT);
```

Build a requester that's connected to the localhost via TCP SERVER_PORT

Overview of the QuotesProxy Class

Overview of the QuotesProxy Class

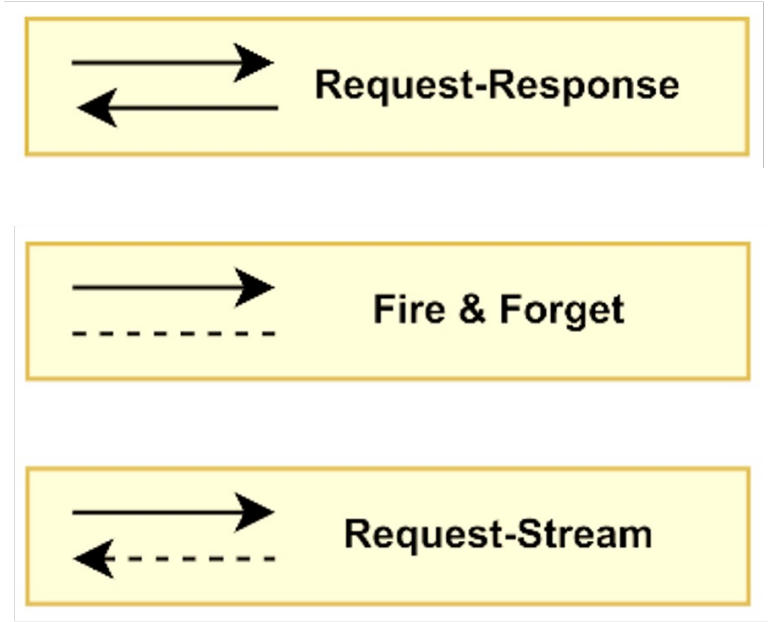
- This proxy defines methods used to send messages to endpoints provided by the QuotesApplication



See RSocket/ex2/src/test/java/quotes/client/QuotesProxy.java

Overview of the QuotesProxy Class

- This proxy defines methods used to send messages to endpoints provided by the QuotesApplication
  - These methods demo each of the three interaction models supported by RSocket



Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created

```
class QuotesProxy {
  @Autowired
  private Mono<RSocketRequester>
    mQuoteRequester;
  ...
```

The RSocketRequester is autowired in QuotesProxy

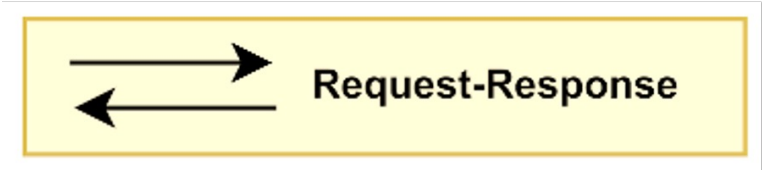
See RSocket/ex2/src/test/java/quotes/client/QuotesProxy.java

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
   (UUID uuid, SubscriptionType t) {
   return mQuoteRequester
      .map(r -> r
           .route(SUBSCRIBE)
           .data(new Subscription
                        (uuid, t)))

      .flatMap(r -> r
                .retrieveMono
                   (Subscription
                    .class))

      .cache();
}
```

Create & return a Subscription that's confirmed by the server



Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
   (UUID uuid, SubscriptionType t) {
   return mQuoteRequester
      .map(r -> r
           .route(SUBSCRIBE)
           .data(new Subscription
                        (uuid, t)))

      .flatMap(r -> r
               .retrieveMono
                  (Subscription
                   .class))

      .cache();
}
```

Create metadata for a message containing data that will be sent to the server to subscribe for quotes

See springframework/messaging/rsocket/RSocketRequester.html#route

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
   (UUID uuid, SubscriptionType t) {
   return mQuoteRequester
      .map(r -> r
           .route(SUBSCRIBE)
           .data(new Subscription
                        (uuid, t)))

      .flatMap(r -> r
               .retrieveMono
                  (Subscription
                   .class))

      .cache();
}
```

Create data for a message containing a subscription id

See springframework/messaging/rsocket/RSocketRequester.RequestSpec.html#data

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
   (UUID uuid, SubscriptionType t) {
   return mQuoteRequester
      .map(r -> r
           .route(SUBSCRIBE)
           .data(new Subscription
                        (uuid, t)))

      .flatMap(r -> r
                .retrieveMono
                  (Subscription
                   .class))

      .cache();
}
```

Send a message to the server to subscribe w/ the given subscription & return a confirmed Subscription as a Mono<Subscription>

See springframework/messaging/rsocket/RSocketRequester.RetrieveSpec.html#retrieveMono

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
   (UUID uuid, SubscriptionType t) {
   return mQuoteRequester
      .map(r -> r
           .route(SUBSCRIBE)
           .data(new Subscription
                        (uuid, t)))

      .flatMap(r -> r
                .retrieveMono
                   (Subscription
                    .class))

      .cache();
}
```

Project Reactor operators are applied on Mono & Flux types to initiate & handle two-way message passing

See spring.io/blog/2016/04/19/understanding-reactive-types

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method

```
Mono<Subscription> subscribe
   (UUID uuid, SubscriptionType t) {
   return mQuoteRequester
      .map(r -> r
           .route(SUBSCRIBE)
           .data(new Subscription
                        (uuid, t)))

      .flatMap(r -> r
                .retrieveMono
                   (Subscription
                    .class))

      .cache();
}
```

Turn this Mono into a hot source & cache last emitted signals for future subscribers

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#cache

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getAllQuotes() proxy method

```
Flux<Quote> getAllQuotes
   (Mono<Subscription> subReq)  {
   return mQuoteRequester
      .zipWith(subReq)

      .flatMapMany(QuotesProxy::
                   getAllQuotes)

      .switchIfEmpty(Flux.error(new
          IllegalAccessException()));
}
```

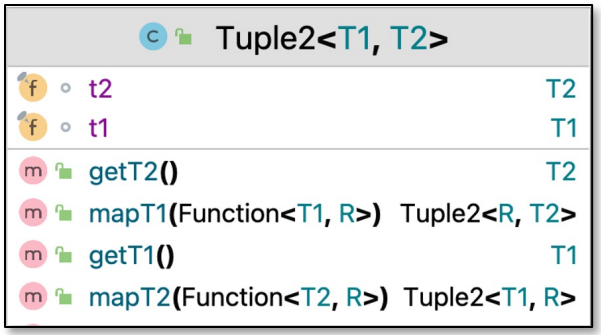
Returns a Flux that emits all the Quotes of a particular type associated with a Subscription

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getAllQuotes() proxy method

```
Flux<Quote> getAllQuotes
   (Mono<Subscription> subReq)  {
   return mQuoteRequester
      .zipWith(subReq)

      .flatMapMany(QuotesProxy::
                   getAllQuotes)

      .switchIfEmpty(Flux.error(new
          IllegalAccessException()));
}
```

Combine the result from this Mono & the other Mono into a Tuple2



See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#zipWith

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getAllQuotes() proxy method

```
Flux<Quote> getAllQuotes
   (Mono<Subscription> subReq)  {
   return mQuoteRequester
      .zipWith(subReq)

      .flatMapMany(QuotesProxy::
                      getAllQuotes)

      .switchIfEmpty(Flux.error(new
          IllegalAccessException()));
}
```

Use Tuple2 metadata & data to pass a message to the server & then convert the response to a Flux that emits a stream of Quote objects

See next part of the lesson for the implementation of this getAllQuotes() method

Overview of the QuotesProxy Class

- Message can be sent after an RSocketRequester is created, e.g.
  - The subscribe() proxy method
  - The getAllQuotes() proxy method

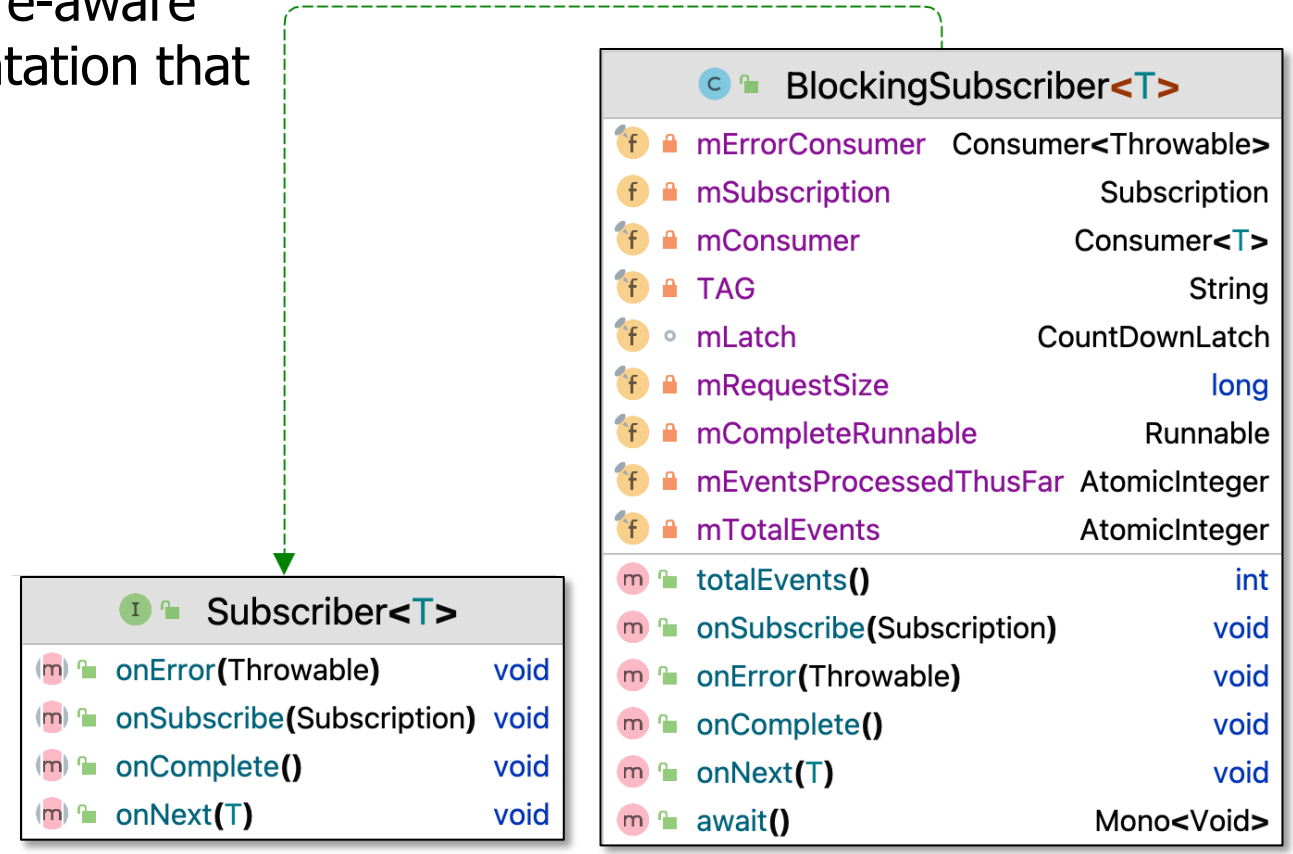
```
Flux<Quote> getAllQuotes
   (Mono<Subscription> subReq)  {
   return mQuoteRequester
      .zipWith(subReq)

      .flatMapMany(QuotesProxy::
                      getAllQuotes)

      .switchIfEmpty(Flux.error(new
          IllegalAccessException()));
}
```

Convert an empty Flux into the IllegalAccessException (which means the client didn't provide a valid subscription)

Overview of the BlockingSubscriber Class

Overview of the BlockingSubscriber Class

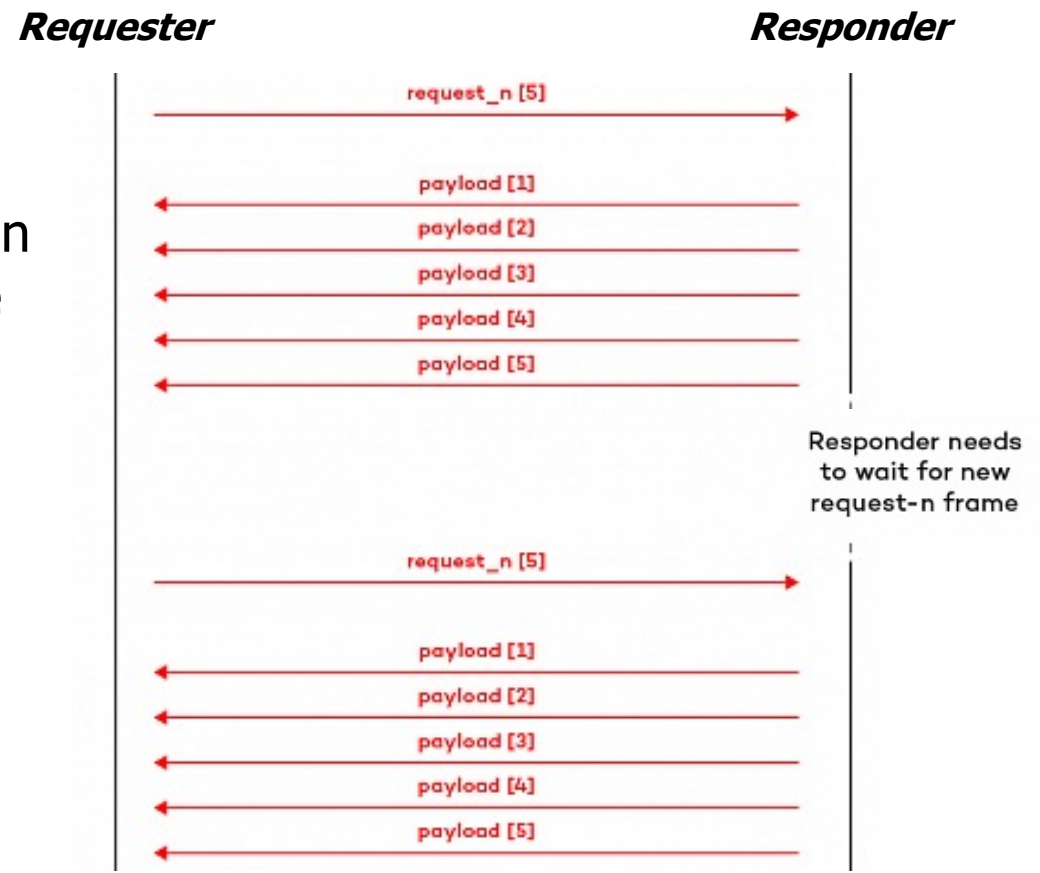
- Define a backpressure-aware Subscriber implementation that handles blocking



See RSocket/ex2/src/test/java/quotes/utils/BlockingSubscriber.java

Overview of the BlockingSubscriber Class

- Define a backpressure-aware Subscriber implementation that handles blocking
  - Uses the Subscriber/Subscription model we covered earlier in the context of Project Reactor



See www.appsdeveloperblog.com/implementing-backpressure-in-project-reactor

End of the Structure & Functionality of the RSocket Quotes Client