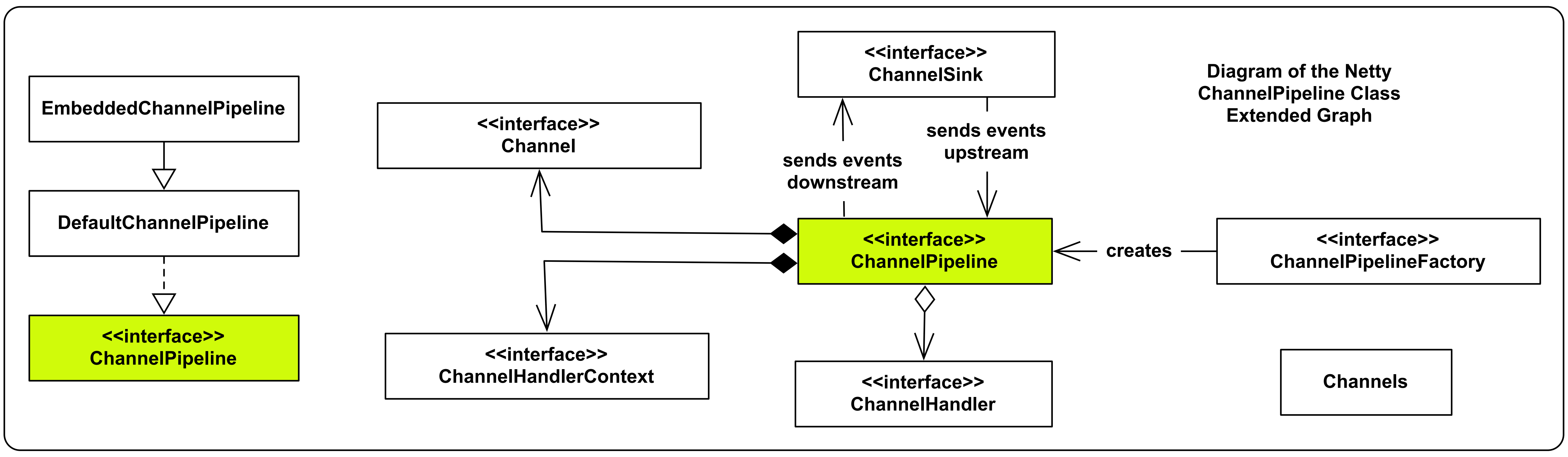

EmbeddedChannelPipeline
DefaultChannelPipeline
<<interface>> ChannelPipeline
<<interface>> Channel
<<interface>> ChannelHandlerContext
<<interface>> ChannelSink
sends events downstream
sends events upstream
<<interface>> ChannelPipeline
<<interface>> ChannelHandler
creates
<<interface>> ChannelPipelineFactory
Channels
Diagram of the Netty ChannelPipeline Class Extended Graph

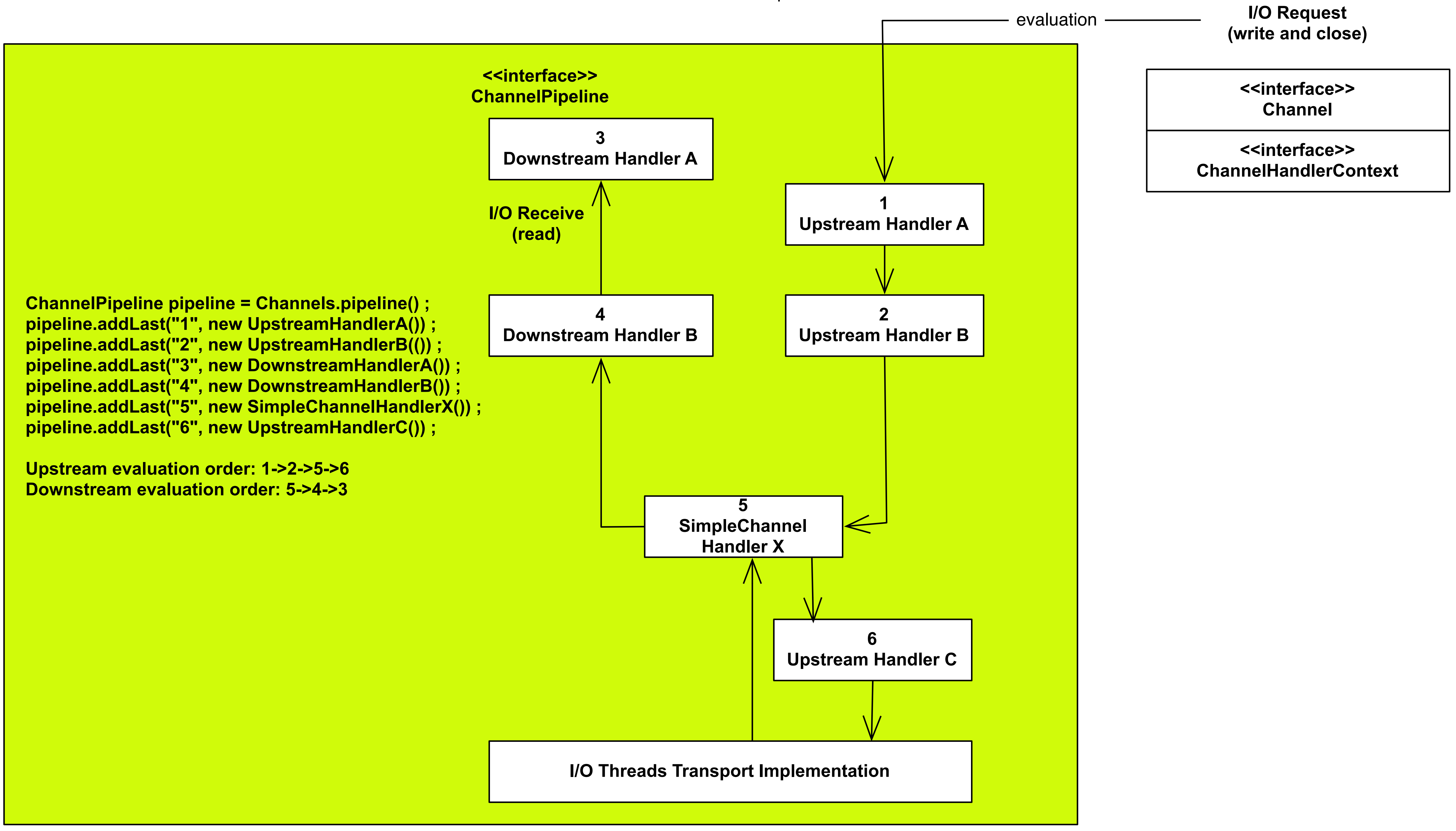
A ChannelPipeline is a list of ChannelHandlers that handle or intercept ChannelEvents of a Channel. ChannelPipeline implements an advanced form of the Interceptino Filter design pattern to give a user of the pipeline full control over how an event is handled and how the ChannelHandlers in the pipeline interact with one another.

Creation

Pipelines should be created using the helper methods in Channels rather than calling implementation constructors.

```
import static org.jboss.netty.channel.Channels.*;
ChannelPipeline pipeline - pipeline() ;  // Or, Channels.pineline() ;
```

Event Flow in a Pipeline


evaluation
I/O Request (write and close)
<<interface>> Channel
<<interface>> ChannelHandlerContext
<<interface>> ChannelPipeline
3 Downstream Handler A
I/O Receive (read)
4 Downstream Handler B
1 Upstream Handler A
2 Upstream Handler B
5 SimpleChannel Handler X
6 Upstream Handler C
I/O Threads Transport Implementation

```
ChannelPipeline pipeline = Channels.pipeline() ;
pipeline.addLast("1", new UpstreamHandlerA()) ;
pipeline.addLast("2", new UpstreamHandlerB(()) ;
pipeline.addLast("3", new DownstreamHandlerA()) ;
pipeline.addLast("4", new DownstreamHandlerB()) ;
pipeline.addLast("5", new SimpleChannelHandlerX()) ;
pipeline.addLast("6", new UpstreamHandlerC()) ;
```

Upstream evaluation order: 1->2->5->6
Downstream evaluation order: 5->4->3

Building a Pipeline

A user will have one or more channel handlers in a pipeline to receive I/O events (e.g. read) and to request I/O operations (e.g. write and close). A typical server will have the following handlers in each channel's pipeline but this varies according to the complexity and characteristics of the protocol and business logic.

1. Protocol Decoder - translates binary data (e.g. ChannelBuffer) into a Java object.
2. Protocol Encoder - translates a Java object into binary data.
3. ExecutionHandler - applies a thread model.
4. Business Logic Handler - performs the actual business logic (e.g. database access).

For example:

```
ChannelPipeline pipeline = Channels.pipeline() ;
pipeline.addLast("decoder", new MyProtocolDecoder()) ;
pipeline.addLast("encoder", new MyProtocolEncoder()) ;
pipeline.addLast("executor", new ExecutionHandler(new OrderedMemoryAwareThreadPoolExecutor(16, 1048576, 1048576) ;
pipeline.addLast("handler", new MyBusinessLogicHandler()) ;
```

Thread Safety

ChannelHandlers can be added and removed at anytime because a ChannelPipeline is thread safe. You can insert an SslHandler when sensitive information is about to be exchanged and remove it after the exchange.

<<interface>>
ChannelPipeline

+addAfter(baseName: String, name: String, handler: ChannelHandler): void
+addBefore(baseName: String, name: String, handler: ChannelHandler): void
+addFirst(name: String, handler: ChannelHandler): void
+addLast(name: String, handler: ChannelHandler): void
+attach(channel: Channel, sink: ChannelSink): void
+get(handlerType: Class<T>): <T extends ChannelHandler> T
+get(name: String): ChannelHandler
+getChannel(): Channel
+getContext(handler: ChannelHandler): ChannelHandlerContext
+getContext(handlerType: Class<? extends ChannelHandler>): ChannelHandlerContext
+getContext(name: String): ChannelHandlerContext
+getFirst(): ChannelHandler
+getLast(): ChannelHandler
+getSink(): ChannelSink
+isAttached(): boolean
+remove(handler: ChannelHandler): void
+remove(name: String): ChannelHandler
+removeFirst(): ChannelHandler
+removeLast(): ChannelHandler
+replace(oldHandler: ChannelHandler, newName: String, newHandler: ChannelHandler): void
+replace(oldHandlerType: Class<T>, new Name: String, newHandler: ChannelHandler): T <T extends ChannelHandler>
+replace(oldName: String, newName: String, newHandler: ChannelHandler): ChannelHandler
+sendDownstream(e: ChannelEvent): void
+sendUpstream(e: ChannelEvent): void
toMap(): Map<String,ChannelHandler>