



UNIVERSIDAD DE BUENOS AIRES  
FACULTAD DE INGENIERÍA  
Año 2015 - 2<sup>do</sup> Cuatrimestre

## **TALLER DE PROGRAMACIÓN III (75.61)**

TRABAJO PRÁCTICO N.º 2  
TEMA: Colas en RabbitMQ  
FECHA: 15/10/2015

Autor  
Darío Eduardo Ramos

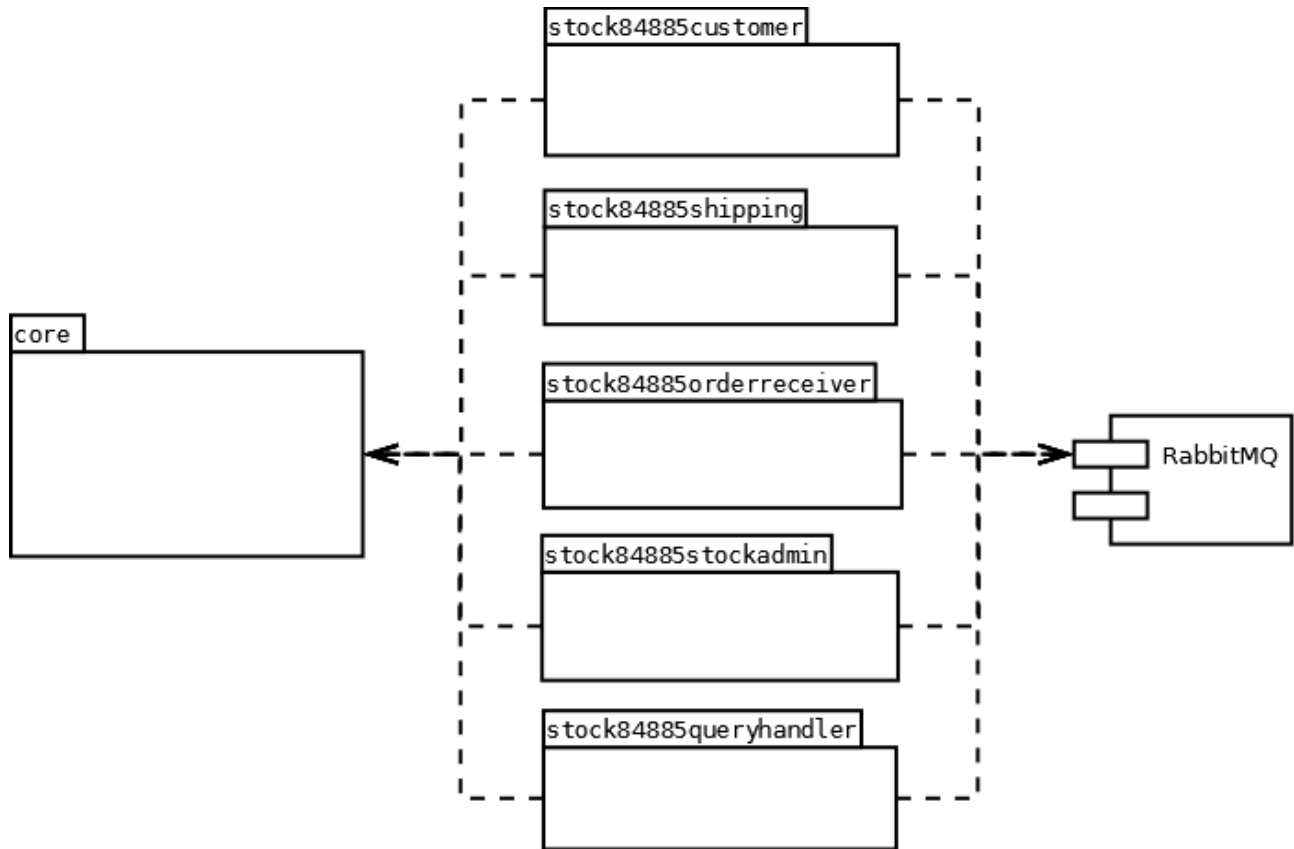
Padrón  
84885

## **Objetivos del sistema**

- Tomar pedidos de una cantidad masiva de usuarios simultáneos (escalabilidad).
- Mantener un control de stock consistente (siempre mayor a cero) en todo momento.
- Diseñar los componentes de manera que puedan ser fácilmente distribuidos.
- Mantener un control detallado de todos los pedidos, con fines de auditoría.
- Utilizar un patrón de comunicaciones persistentes para garantizar el envío de los mensajes.

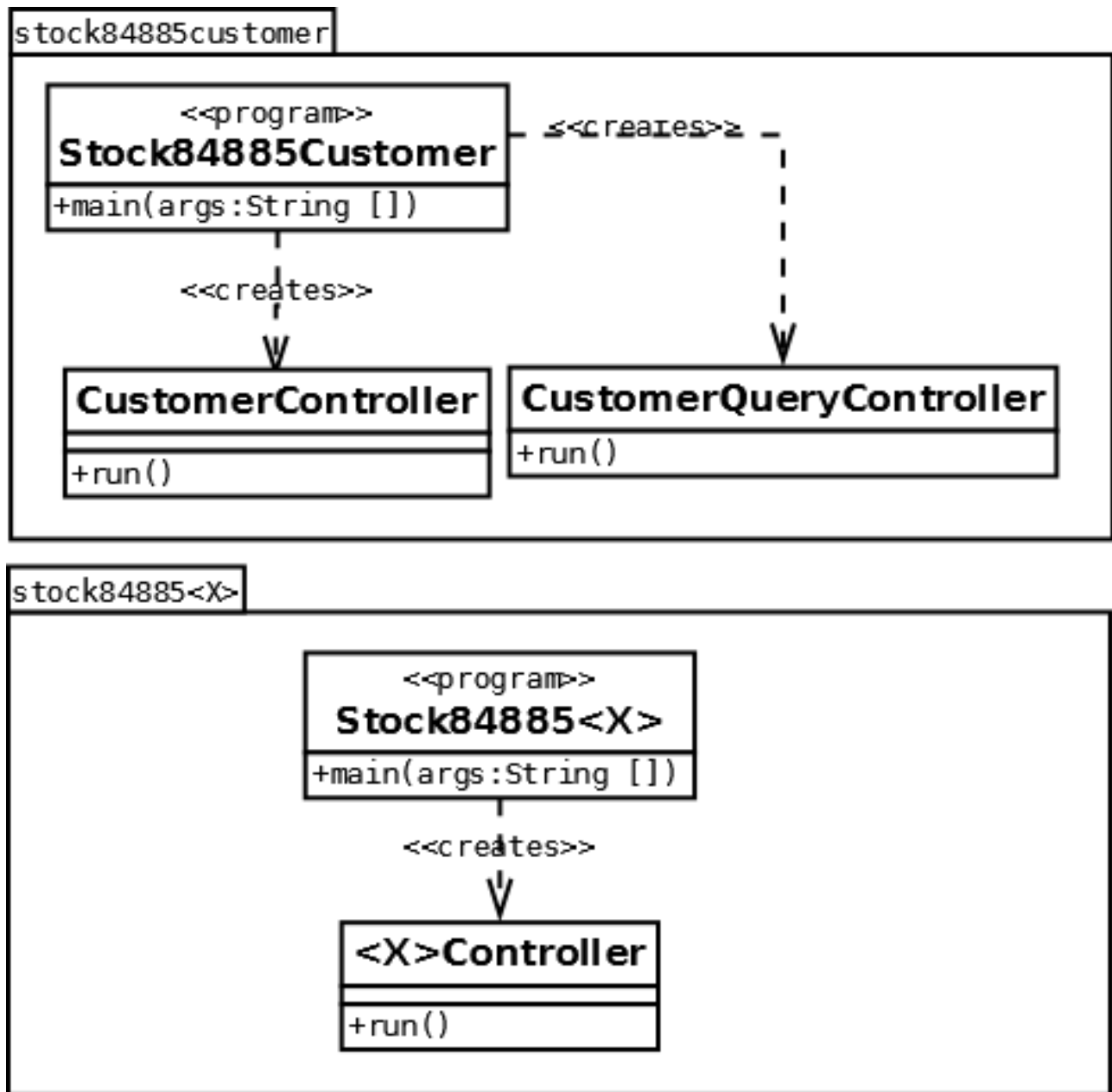
## Vista lógica: Diagrama de paquetes

### Diagrama de paquetes general



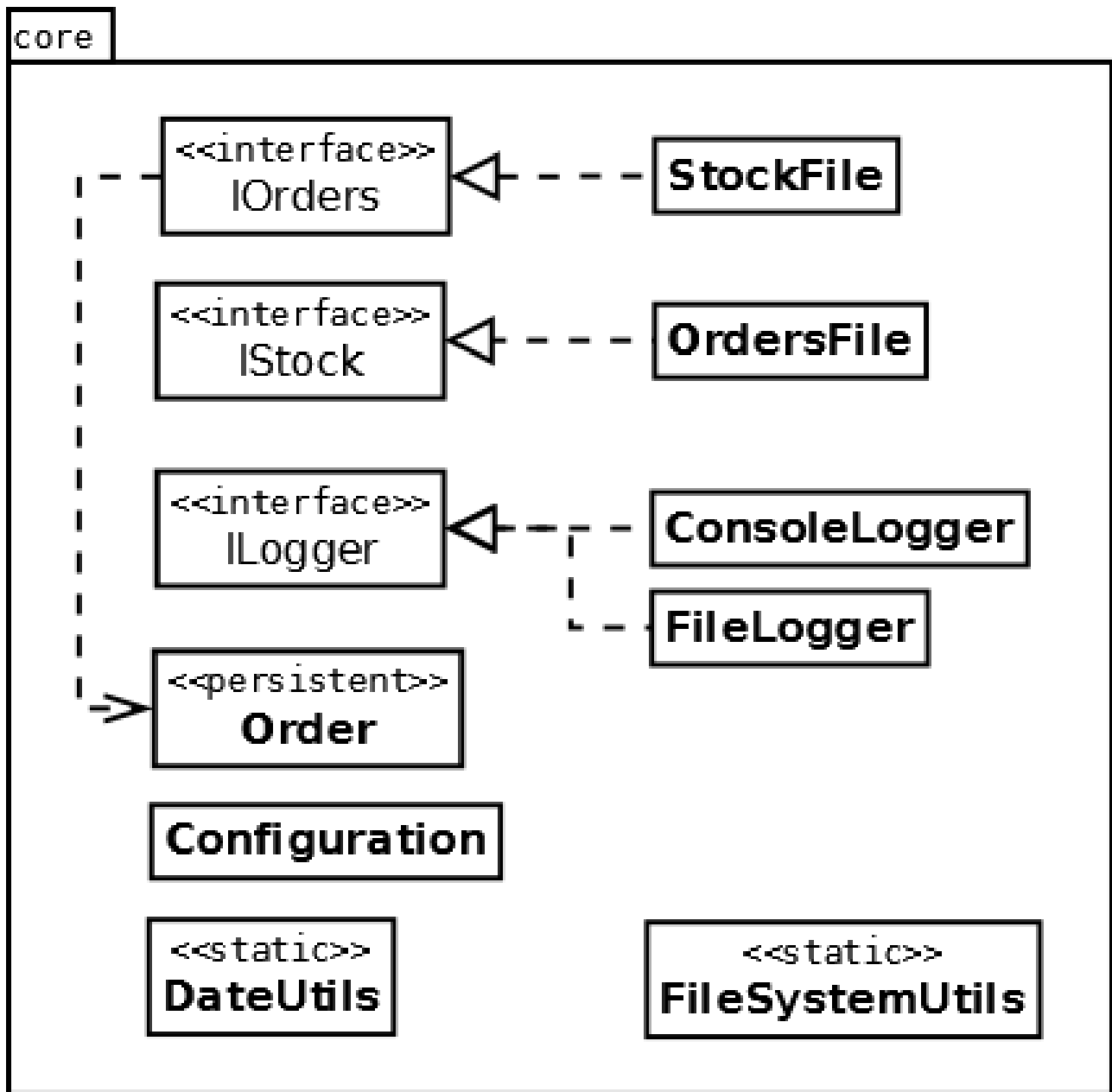
Todos los paquetes del centro están asociados a un ejecutable (.jar), usan RabbitMQ, y clases del Core que no se muestran en este diagrama por simplicidad.

## Diagrama de paquetes y clases (ejecutables)



Todos los ejecutables tienen la misma estructura de clases, excepto **stock84885Customer**, que tiene dos controllers, **CustomerController** y **CustomerQueryController**. Todos los otros, simbolizados por una X en el segundo paquete "genérico" (X = **stock84885shipping**, **stock84885orderreceiver**, **stock84885stockadmin** y **stock84885queryhandler**), constan de un ejecutable y un Controller.

## Diagrama de clases del paquete stock84885core



Las interfaces `Iorders` e `Istock` abstraen de la implementación de la persistencia de Órdenes (clase `Order`) y el `Stock`; las implementaciones provistas, `StockFile` y `OrdersFile`, son archivos accedidos con exclusión mutua entre procesos usando un `FileLock`.

La interfaz `Ilogger` abstrae al usuario del medio de logging (archivo, consola, socket, etc).

`Configuration` hace transparente el acceso a los parámetros del sistema.

`DateUtils` y `FileSystemUtils` son clases auxiliares, sin estado (por ende el estereotipo `static`).

## Descripción de clases más importantes

Clase	Descripción
<b>StockFile</b>	<b>Responsabilidad:</b> Almacenar el stock actual garantizando consulta y actualización en exclusión mutua.
	<b>Atributos principales:</b>  <u>_maxStock</u> : Máximo valor de stock para cualquier tipo de producto.  <u>_filePath</u> : Ruta del archivo de stock.  <u>_lockFilePath</u> : Ruta del archivo de lock asociado al archivo de stock (podría ser el mismo si se leyera el mismo de forma binaria).
	<b>Métodos principales:</b>  <i>public boolean decrement(Order.EProductType type, int count)</i> : Devuelve true sii se pudo reducir el stock del tipo y cantidad especificados. Exclusión mutua garantizada.  <i>public boolean increment(Order.EProductType type, int count)</i> : Análogo a increment, pero aumentando stock. Se valida que no supere el máximo valor.
<b>OrdersFile</b>	<b>Responsabilidad:</b> Almacenar los pedidos (orders) garantizando consulta y creación en exclusión mutua.
	<b>Atributos principales:</b>  <u>_filePath</u> : Ruta del archivo de pedidos.  <u>_lockFilePath</u> : Ruta del archivo de lock asociado al archivo de pedidos (podría ser el mismo si se leyera el mismo de forma binaria).
	<b>Métodos principales:</b>  <i>public List&lt;Order&gt; getOrdersByUsername(String userName)</i> : Devuelve la lista completa de pedidos realizados por el usuario de nombre userName. Si nunca hizo un pedido, se devuelve una lista vacía. La consulta se hace en exclusión mutua.  <i>public void create(Order order, Order.EOrderState initialState)</i> : Crea un pedido con los datos contenidos en order, y el estado de pedido inicial especificado por initialState.  <i>public void setState(Order order, Order.EOrderState state)</i> : Altera el estado de order, garantizando exclusión mutua.  <i>public void setState(String orderID, Order.EOrderState state)</i> :

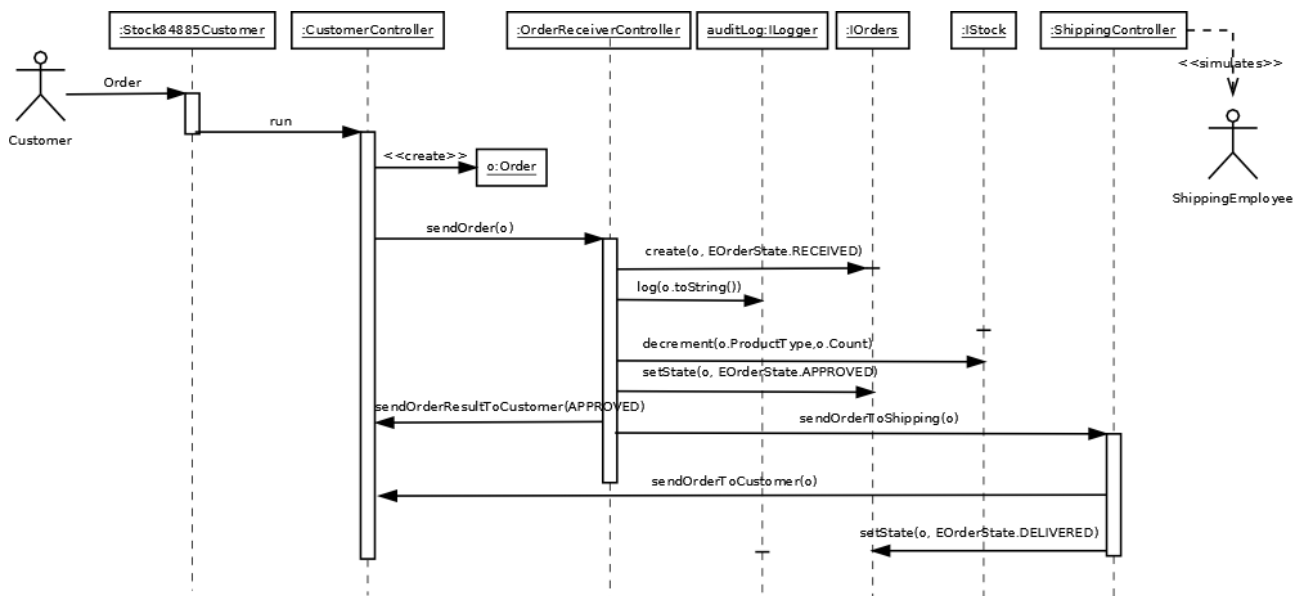
	<p>Overload para mayor comodidad. Busca el pedido con orderID dado, y actualiza su estado garantizando exclusión mutua. Si dicho pedido no existe, lanza una excepción.</p>
Order	<p><b>Responsabilidad:</b> Representar un pedido, con sus valores y estado dentro del sistema (recibido, aprobado, rechazado o enviado).</p>
	<p><b>Atributos principales:</b></p> <p><i>CustomerName:</i> Nombre del cliente que realizó el pedido.</p> <p><i>ProductType:</i> Tipo de producto encargado en el pedido.</p> <p><i>_id:</i> Identificador unívoco del pedido dentro del sistema.</p> <p><i>Count:</i> Cantidad de unidades de producto de tipo ProductType encargados.</p> <p><i>State:</i> Estado actual del pedido: recibido, aprobado, rechazado o enviado. Como los pedidos son creados al ser recibido, el estado inicial siempre es recibido.</p>
	<p><b>Métodos principales:</b></p> <p><i>public byte[] serialize():</i> Devuelve una representación binaria del pedido.</p> <p><i>public static Order deserialize(byte[] bytes):</i> A partir de una representación binaria creada por serialize, se obtiene un objeto equivalente al original.</p>
CustomerController	<p><b>Responsabilidad:</b> Simular a un cliente, enviando un pedido generado aleatoriamente a un OrderReceiver, y recibiendo la respuesta al mismo.</p>
	<p><b>Atributos principales:</b></p> <p><i>_deliveryChannel:</i> Canal para recibir la entrega del pedido.</p> <p><i>_resultChannel:</i> Canal para recibir el resultado del pedido (aprobado o rechazado). Se recibe usando _name como routing key.</p> <p><i>_name:</i> Nombre del cliente en el sistema. Sirve para identificarlo al enviarle el resultado o entrega.</p>
	<p><b>Métodos principales:</b></p> <p><i>public void run():</i> Inicializa los canales, genera un pedido aleatorio, lo envía y recibe la respuesta.</p>
CustomerQueryController	<p><b>Responsabilidad:</b> Simular a un cliente que consulta el estado de su pedido.</p>

	<p><b>Atributos principales:</b></p> <p><i>_queryChannel</i>: Canal para enviar la consulta.</p> <p><i>_queryResultChannel</i>: Canal para recibir el resultado de la consulta. Se usa <i>_name</i> como routing key.</p> <p><i>_name</i>: Nombre del cliente en el sistema. Sirve para identificarlo al enviarle el resultado.</p> <p><b>Métodos principales:</b></p> <p><i>public void run()</i>: Inicializa los canales, genera una consulta aleatoria, la envía y espera la respuesta.</p>
<b>OrderReceiverController</b>	<p><b>Responsabilidad:</b> Procesar pedidos, actualizando el stock pertinentemente y despachando pedidos a Shipping. <b>Registrar pedidos nuevos en el log de auditoría.</b></p> <p><b>Atributos principales:</b></p> <p><i>_ordersChannel</i>: Canal para recibir los pedidos.</p> <p><i>_resultsChannel</i>: Canal para enviar los resultados a cada cliente. Se usa el nombre del cliente como routing key.</p> <p><i>_shippingChannel</i>: Canal para enviar pedidos a Shipping.</p> <p><i>_auditLogger</i>: Para registrar hora de creación de cada pedido.</p> <p><i>_orders</i>: Repositorio de pedidos. Se accede para crearlos y actualizar su estado (aprobado o rechazado).</p> <p><i>_stock</i>: Archivo de stock. Se accede para actualizarlo luego de aprobar un pedido.</p> <p><b>Métodos principales:</b></p> <p><i>public void run()</i>: Inicializa los canales y entra en un loop donde recibe y procesa pedidos. Si el pedido no es rechazado, se actualiza el stock, se registra el pedido en el repositorio de pedidos y el log de auditoría, y luego es enviado a shipping. El procesamiento es secuencial (un pedido a la vez).</p>
<b>QueryHandlerController</b>	<p><b>Responsabilidad:</b> Recibir y responder consultas de estado de pedidos.</p> <p><b>Atributos principales:</b></p> <p><i>_queriesChannel</i>: Canal para recibir las consultas.</p> <p><i>_queriesResultChannel</i>: Canal para enviar los resultados de las consultas. Se usa el nombre del cliente como routing key.</p>



	<p><b>Métodos principales:</b></p> <p><i>public void run()</i>: Inicializa los canales y entra en un loop donde recibe y procesa consultas. Si el cliente que consulta nunca hizo un pedido, se le devuelve una lista de pedidos vacía.</p>
<b>ShippingController</b>	<p><b>Responsabilidad:</b> Simular al actor Empleado de despacho, recibiendo pedidos y enviándolos al cliente que los realizó.</p>
	<p><b>Atributos principales:</b></p> <p><i>_shippingChannel</i>: Canal para recibir los pedidos a reenviar.</p> <p><i>_deliveryChannel</i>: Canal para enviar los pedidos. Se usa el nombre del cliente como routing key; el mismo está contenido en el pedido mismo.</p> <p><i>_orders</i>: Repositorio de pedidos. Se accede para marcar el pedido como enviado una vez hecha tal cosa.</p>
	<p><b>Métodos principales:</b></p> <p><i>public void run()</i>: Inicializa los canales y entra en un loop donde recibe y procesa pedidos.</p>
<b>StockAdminController</b>	<p><b>Responsabilidad:</b> Actualizar stock.</p>
	<p><b>Atributos principales:</b></p> <p><i>_stock</i>: Para actualizar el stock garantizando exclusión mutua.</p> <p><i>_maxProductCount</i>: Máximo delta de stock (positivo o negativo). Es configurable.</p>
	<p><b>Métodos principales:</b></p> <p><i>public void run()</i>: Genera un tipo de producto y un delta aleatorios e intenta actualizar el stock con los mismos.</p>

## Vista de procesos: Diagrama de secuencia



Este diagrama corresponde al escenario más común: un cliente hace un pedido, hay stock suficiente y se le entrega el pedido sin problemas. Se omitieron los mecanismos de comunicación por simplicidad; los mismos pueden verse en la vista de despliegue (diagrama de robustez).

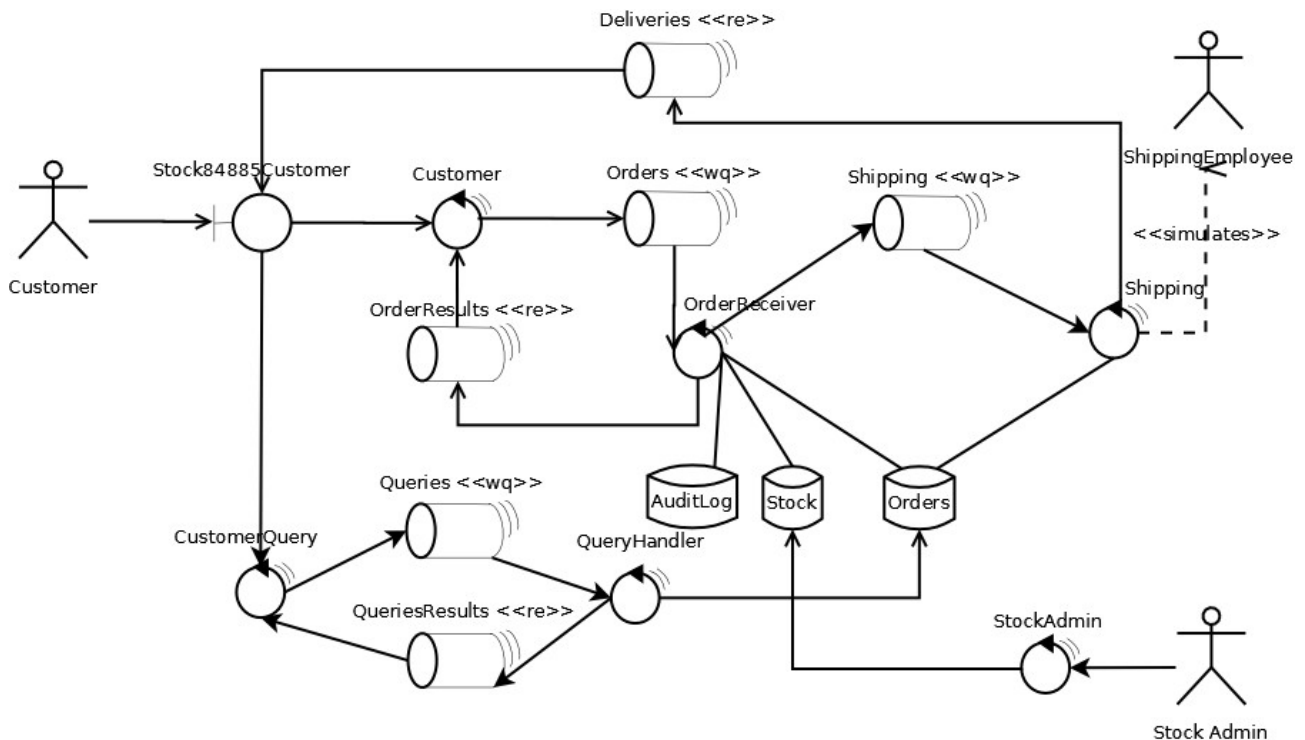
El cliente (Customer) manda su pedido a través de un objeto de borde de tipo **Stock84885Customer**, quien invoca a **CustomerController**. Éste instancia una **Order**, o, con los datos del pedido, y se la envía a **OrderReceiverController**. El OrderReceiver registra la orden con estado inicial RECEIVED, y **registra el nuevo pedido en el log de auditoría (auditLog en el diagram)**. Luego, intenta decrementar el stock según el pedido. En este escenario, lo logra, y por ende cambia el estado del pedido a APPROVED, notifica al cliente que su pedido fue aprobado y envía la orden a un **ShippingController**.

En Shipping, la orden es despachada al cliente que la realizó, se marca la orden como enviada (DELIVERED), y se cierra el circuito.

El diagrama para el caso en que el pedido es rechazado sería similar, con la diferencia de que se marca la orden como rechazada (REJECTED), se notifica al cliente y se corta el circuito allí.

El acceso a IOrders e IStock es en exclusión mutua para toda objeto que los acceda, sea para leer o escribir.

## Vista de despliegue: Diagrama de robustez



Estereotipos:

- re: Routing Exchange. Usa routing para enviar el mensaje a un receptor específico. Por ejemplo, Shipping lo usa para enviarle el pedido al cliente que lo pidió y a ningún otro.
- wq: Workers Queue. El productor pone el mensaje en el exchange, y lo toma cualquier consumidor (se usa el esquema por defecto, o sea Round Robin). Por ejemplo, lo usa el Customer para enviar su pedido y que lo tome cualquier OrderReceiver.

Sobre el diagrama:

Un cliente (**Customer**) envía un pedido (order) por la cola **Orders**, de donde la toma algún **OrderReceiver**. El mismo crea una instancia de Order y la guarda en **Orders** con estado inicial recibido. **También registra el nuevo pedido en el log de auditoría**. Luego, revisa el **Stock**, garantizando exclusión mutua, y le responde al **Customer** si el pedido fue rechazado porque no había stock, o aprobado. En ambos casos, se actualiza el estado del pedido.

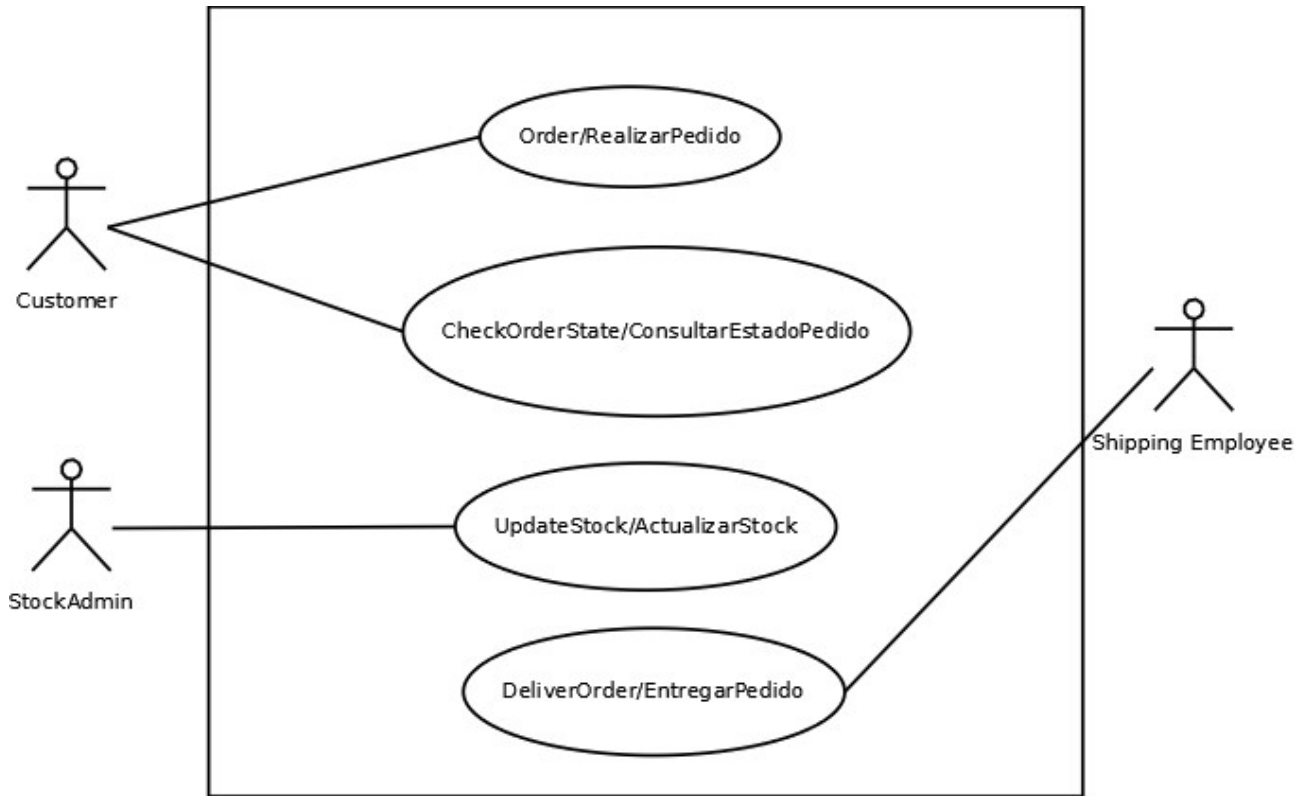
Si el pedido es aprobado, es enviado a la cola **Shipping**, de donde la toma algún **ShippingController** y la envía por el exchange **Deliveries** al cliente que la pidió. Shipping sabe quién es ese cliente porque dicha información llegó en el pedido que recibió.

Por otro lado, un Customer puede consultar el estado de sus pedidos. Lo hace a través de **CustomerQuery**, quien envía el nombre del cliente por **Queries**, de donde lo toma **QueryHandler**. Éste revisa Orders, y manda la lista de pedidos con su estado al cliente vía **QueriesResults**.

Por último, el actor StockAdmin, a través del **controller homónimo**, puede consultar el stock de cualquier producto en cualquier momento, con exclusión mutua garantizada.

## Vista de casos de uso

### Diagrama de casos de uso



## Especificación de casos de uso

<b>Caso de uso</b>	RealizarPedido
<b>Actores</b>	Primario: Cliente (Customer) Secundario: Empleado de Despacho (Shipping Employee)
<b>Pre-condiciones</b>	Stock definido y válido
<b>Descripción</b>	<p>Flujo principal</p> <ol style="list-style-type: none"> <li>1. Cliente envía su pedido especificando tipo de producto y cantidad.</li> <li>2. Sistema registra pedido como recibido.</li> <li>3. Sistema registra pedido en log de auditoría.</li> <li>4. Sistema consulta stock de producto y cantidad solicitados.</li> <li>5. Sistema marca pedido como aprobado.</li> <li>6. Sistema responde a Cliente que su pedido fue aprobado.</li> <li>7. Sistema envía pedido a Empleado de Despacho</li> </ol> <p>Flujo alternativo A1 (si no hay stock para el pedido)</p> <p>A1.1. Sistema marca pedido como rechazado.</p> <p>A1.2. Sistema responde a Cliente que su pedido fue rechazado.</p>
<b>Post-condiciones</b>	Stock actualizado (decrementado) Pedido registrado en el sistema como rechazado o enviado

<b>Caso de uso</b>	ConsultarEstadoPedido
<b>Actores</b>	Primario: Cliente
<b>Pre-condiciones</b>	Registro de pedidos activo y válido
<b>Descripción</b>	<p>Flujo principal:</p> <ol style="list-style-type: none"> <li>1. Cliente solicita a Sistema el estado de su(s) pedido(s).</li> <li>2. Sistema devuelve el listado de todos los pedidos realizados por Cliente, incluyendo su estado actual al momento de la consulta.</li> </ol> <p>Flujo alternativo A1:</p> <p>(si el cliente nunca hizo un pedido)</p> <p>A1.1. Sistema responde que no hay pedidos registrados</p>
<b>Post-condiciones</b>	El Cliente tiene la información solicitada. No hay cambios en el Sistema.

<b>Caso de uso</b>	ActualizarStock
<b>Actores</b>	Primario: Administrador de Stock (StockAdmin)
<b>Pre-condiciones</b>	Registro de stock válido
<b>Descripción</b>	<p>Flujo principal:</p> <ol style="list-style-type: none"> <li>1. Admin solicita un cambio en el stock, especificando tipo de producto y variación en stock (positiva o negativa).</li> <li>2. Sistema realiza modificación.</li> </ol> <p>Flujo alternativo A1:</p> <p>(no se puede hacer el cambio porque daría stock negativo o por encima del máximo)</p> <p>A1.1. Sistema informa a Admin que no se puede efectuar el cambio</p>
<b>Post-condiciones</b>	Stock actualizado.

<b>Caso de uso</b>	EntregarPedido
<b>Actores</b>	Primario: Empleado de Despacho (Shipping Employee) Secundario: Cliente (Customer)
<b>Pre-condiciones</b>	Empleado listo para recibir pedidos Cliente esperando entrega de pedido.
<b>Descripción</b>	<p>Flujo principal:</p> <ol style="list-style-type: none"> <li>1. Empleado espera pedidos.</li> <li>2. Sistema envía un pedido.</li> <li>3. Empleado determina a qué cliente entregar el pedido.</li> <li>3. Empleado envía pedido a Cliente.</li> <li>4. Empleado marca pedido como enviado.</li> </ol>
<b>Post-condiciones</b>	Cliente recibió su pedido.