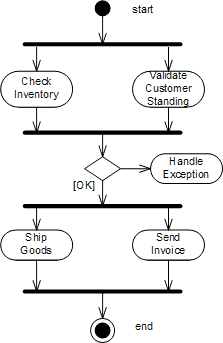
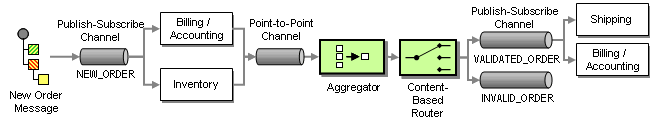
**Exercise 1**

Use RabbitMQ to implement a messaging solution for an Order system as described in [chapter 1](https://www.enterpriseintegrationpatterns.com/patterns/messaging/Chapter1.html) of the Enterprise Integration Pattern book (see where to run RabbitMQ in the end of the document). The solution must be designed as a [Pipes and Filters architecture](https://www.enterpriseintegrationpatterns.com/patterns/messaging/PipesAndFilters.html). All the filters are business process modules, such as “Check Inventory” or “Ship Goods” that can be simple business simulations. You can decide your own client type that starts the process (e.g. console app or web app). The client process does not have to get a reply back. The reply can go into a database or email (or a simulation ☺).

From a business point of view, the workflow of the ordering process looks like this:



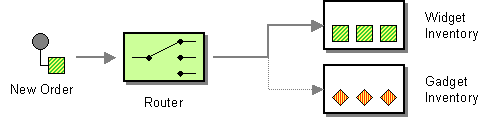
You must use the integration design illustrated below. There must be a [*Publish-Subscribe Channel*](https://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html) to implement the fork action between Billing/Accounting and Inventory plus an [*Aggregator*](https://www.enterpriseintegrationpatterns.com/patterns/messaging/Aggregator.html) to implement the join action. A *Publish-Subscribe Channel* sends a message to all active consumers while an *Aggregator* receives multiple incoming messages and combines them into a single outgoing message (see picture):



The resulting message from the aggregator is passed to a [*Content-Based Router*](https://www.enterpriseintegrationpatterns.com/patterns/messaging/ContentBasedRouter.html). A Content-Based-Router is a component that consumes a message and publishes it unmodified to a choice of other channels based on rules coded inside the router. In this case, if both the inventory check and the credit check have been affirmative, the *Content-Based Router* forwards the message to the VALIDATED\_ORDER channel. If the customer is not in good standing or we have no inventory on hand, it forwards the message to the INVALID\_ORDER process. The exception process (not shown in the picture) listens to messages on this channel and notifies the customer of the rejected order.

**Exercise 2**

Now, there are extra challenges due to a business merge of two companies, one for Widgets and one for Gadgets. The company therefore has two inventory systems, one for widgets and one for gadgets. As a result, you have to route the request for inventory to the correct system. You can use a RabbitMQ direct exchange type with a routing key to solve that:



**RabbitMQ broker – which to use?**

You can use a RabbitMQ broker that Cphbusiness provides at datdb.cphbusiness.dk.

You can access it from code like this (Java example):

ConnectionFactory factory = new ConnectionFactory();

factory.setHost("datdb.cphbusiness.dk");

factory.setPort(5672);

factory.setUsername("student");

factory.setPassword("cph");

You can access the management web interface here: <http://datdb.cphbusiness.dk:15672/>

Alternatively use an AMQP cloud service at <https://www.cloudamqp.com/plans.html>.

Little Lemur is a shared instance for hobby projects where you have a monthly message limit of 1M. So do not use it for high activity tasks such as testing performance, scalability etc. You can see Java demo code of how to access a cloud instance here: <https://github.com/Tine-m/Rabbit-cloud>