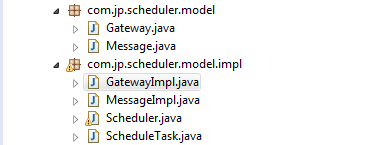
**Notes of technical decisions**

First of all we are going to identify the main interfaces and classes: ‘**Gateway’** and ‘**Message’** Interfaces. The implementation classes **GatewayImpl** and **MessageImpl**.

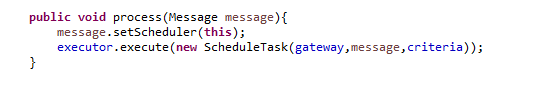
We also have to implement the ‘**Scheduler’** class, it will be the main class of our system.   


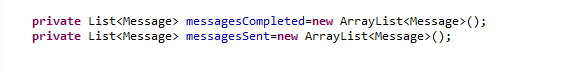
* **We want to make sure that they are not idle when messages are waiting to be processed.**
* **When no resources are available, messages should not be sent to the Gateway**

This is common pattern implemented by the JSE *ThreadPoolExecutor* API (https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html).  
The task to be executed will represent a request to the Gateway interface.  
  
We are going to add a reference to a ThreadPoolExecutor object in the Scheduler class. The ThreadPoolExecutor will execute the thread, if the thread cannot be executed, then the task is added to the queue. When there are threads available, the executor automatically takes the next Thread to be executed from the queue.

The Schedule class will be the responsible to send messages throw the Gateway interface. We implement this action in ‘**ScheduleTask’** thread.

* **As messages are completed, if there are queued messages, they should be processed.**

We are going to declare a reference to the Scheduler object in the Message class. When the message is scheduled we call the method in this way (***message.setSchedule(this))*** to allow the Scheduler to be notified when a message is sent or responded.   
  
The Scheduler object will handle the Message events and register them in the list messagesCompleted and messagesSent in order of execution.



* **Where possible, the message groups should not be interleaved...except where resources are idle and other work can be done.**
* **The priority in which to process groups is defined by the order in which you receive the first message from the group**
* **If there are messages belonging to multiple groups in the queue, as resources become available, we want to prioritise messages from groupsalready started**

*To fit the selection requirements, w*e will use the PriorityBlockingQueue for the ThreadPoolExecutor.   
Then the ScheduleTask must implement the compare interface.   
  
The Gateway will register the order of groups in a HashMap when new groups are processed. It will be necessary to check the priority.   
  
We will have to implement different selection strategies, so we are going to create an abstract class and some implementation class for each strategy. In the ScheduleTask we will add an attribute as a reference to the abstract class ScheduleAlgorithm. The ScheduleAlgorithm will do the comparation work for the ScheduleTask.

We will implement two different strategies for selection. The classes DefaultAlgorithm (wich priorizes groups being processed ) and AlternativeAlgorithm.   
The AlternativeAlgorithm will implement the opposite behavior of DefaultAlgorithm.

* **It should be possible to tell the scheduler that a group of messages has now been cancelled.**

We are going to implement a validation method in the GatewayImpl class.  
This method will be executed by the ScheduleTask before we send the message.   
  
In this method we could implement several criteria to discard the message or send it. We will ask here for cancellation of groups.

**Test Cases**

@Test

**public** **void** testSingleResource()

Request Order:   
Message1(1), Message2(1), Message3(1), Message4(2), Message5(2), Message6(3), Message7(3), Message8(2), Message9(2), Message10(1)

Expected Sent Order:   
Message1(1), Message2(1), Message3(1), Message10(1), Message4(2),Message5(2), Message8(2), Message9(2), Message6(3), Message7(3)

@Test

**public** **void** testSeveralResources()

Test with 2 resources

Request Order:   
Message1(1), Message2(1), Message3(1), Message4(2), Message5(2), Message6(3), Message7(3), Message8(2), Message9(2), Message10(1)

Expected sent order:   
Two last sent messages should be of group 3

@Test

**public** **void** testCancellation()

Request Order:

Message1(1), Message2(1), Message3(1), Message4(2), Message5(2), Message6(3), Message7(3), Message8(2), Message9(2), Message10(1), cancel(2)

Expected sent messages:   
Messages of group 2 should not be sent

@Test

**public** **void** testAlternativeAlgorithm()

Request Order:

Message1(1), Message2(1), Message3(1), Message4(1), Message5(1), Message6(2), Message7(3), Message8(4), Message9(5), Message10(6)

Expected sent order:

Message 1(1), Message 6(2), Message 7(3), Message 8(4), Message 9(5), Message 10(6), Message 2(1), Message 3(1), Message 4(1), Message 5(1)

@Test

**public** **void** testStressResources()

/\*\*

\* Test with 10 resources and 100 messages (10 messages processed per resource) It should last the same time as testSingleResource \*/