This project can be divided into two main parts:

* The scheduler that receives messages and has to organized them before sending
* The Resource that consumes the messages

**SCHEDULER PART**

His role is to receive messages and send them to the resource via the gateway.

**SCHEDULER OBJECT**

The ***Scheduler*** object is a Thread. Its constructor needs two parameters:   
- The *gateway* to send message to.  
-The Message Prioritisation algorithm. Hence, when we need to change the algorithm, we just have to pass another object to the scheduler. To do so, this algorithm object has to implement an interface (*MessagePrioritisation* interface)

The two main methods of the ***Scheduler*** are  
- *receiveMessage()* from an object. We have created a ***GenerateMessage*** object, as an independent Thread, that sends messages to the Scheduler.   
  
- *sendMessage()* to the gateway. Two important points for this method:  
 -Need to be synchronized with the resource to know when the resource is available to receive a new message (see **ResourceLock**)  
 -Need to send the “correct” message

The *run()* method of this thread is to wait for new messages and send them when the resource is available.

**MESSAGE PRIORITISATION**

The ***MessagePrioritisationImplOriginal*** object implements the algorithm for sending the correct message to the Gateway.  
 **ADD MESSAGE**  
When a message arrives, it checks if its group ID exists in the **LinkedHashMap** . If no, it adds the goup ID as KEY and for the value, it uses a **LinkedList** to store all the messages bellowing to the group.  
If the group ID exists but the **LinkedList** is null, it means that the group has been canceled. The message is not added.  
**LinkedHashMap** is used instead of HashMap in order to preserve the arrival order of the group ID.  
 **GET MESSAGE**  
To get the next correct message, it checks first if the group Id is not in the **TerminationList** (see below). If yes it throws a **TerminationException**.   
Otherwise, it starts from the first element of the **LinkedHashMap** and check if its **linkedList** contains messages. If so, it returns and removes the first item on the **linkedList**. Otherwise it gets the second element of the **LinkedHashMap** and so on.

**CANCELLATION**  
For the cancellation features, the method *cancelGroup* set the **LinkedList** of the group to null. Hence no more messages can be store and retrieve for that group ID

**TERMINATION MESSAGE**When a message contains the termination parameter (*isTerminationMessage*) to true, it add its group ID in the **TerminationList** (when the message is added).

**MESSAGE OBJECT**

The **MessageImpl** object implements the message interface. It has 3 fields:   
- group\_id  
-message\_id  
- isTerminationMessage

**RESOURCE PART**

**THE GATEWAY**

The gateway is the only entry point between the scheduler and the resource. There are 4 methods:

*send(Message msg)* : called by the scheduler to send a message to the resources.  
*waitForFreeResource()* : called by the scheduler to know when a resource in available

*startResource()* and *stopResource()* : called by the *MAIN* method to start and stop the resource.

**RESOURCE MANAGER**

The **resourcemanager** object manages the pool of workers (Thread).  
 It also receives messages from the scheduler via the gateway. When a message arrives, it stores it in the waiting message queue. Hence, a worker can pick it up.

**WAITINGMESSAGE**

This **waitingMessage** object is a singleton. It is share between workers and the resource manager. It is queuing messages. As soon as a worker is free, the worker gets a message form the **waitingMessage** queue. The worker can be waiting until a new message arrives.

When a message is added, with *putMessage()* method , one resource is immediately removed (see below **ResourceLock)** to make sure that a second message won’t be adding if there is only one resource available.

To manage the concurrent access to this queue, methods are synchronized.

**RESOURCELOCK**

This **ResourceLock** object is also a singleton. It is share between the scheduler and the workers. Thanks to the method *waitForFreeResource()* the scheduler waits until a worker is free. It can then send a message to the gateway.   
When a worker has finished its job, it notifies the waiting scheduler.

To manage the concurrent access to this object, methods are synchronized

**WORKER**

The ***Worker*** object is a thread. It performs the job with the message.   
When created, it adds a new resource to the **ResourceLock** object. It notifies the **Scheduler** that might be waiting for a free resource.  
Then it is waiting for an available message to treat. Once the treatment is done, the worker adds a new resource and notifies the scheduler.