**Question 1:** What is the code part that shows that the Server assigns the correlation ID to the response?

Answer:

DeliverCallback deliverCallback = (consumerTag, delivery) -> {  
 AMQP.BasicProperties replyProps = new AMQP.BasicProperties  
 .Builder()  
 **.correlationId(delivery.getProperties().getCorrelationId())** .build();

**Question 2:** You base on both code of Client and Server program to explain which code shows that the Client sends request to Server through rpc\_queue and create a new queue to wait for the reply of the Server.

Answer:

* In Client side:

public String call(String message) throws IOException, InterruptedException {  
 final String corrId = UUID.*randomUUID*().toString();  
 String replyQueueName = channel.queueDeclare().getQueue();  
 AMQP.BasicProperties props = new AMQP.BasicProperties  
 .Builder()  
 .correlationId(corrId)  
 .replyTo(replyQueueName)  
 .build();  
 channel.basicPublish("", requestQueueName, props, message.getBytes("UTF-8"));  
 final BlockingQueue<String> response = new ArrayBlockingQueue<>(1);  
 String ctag = channel.basicConsume(replyQueueName, true, (consumerTag, delivery) -> {  
 if (delivery.getProperties().getCorrelationId().equals(corrId)) {  
 response.offer(new String(delivery.getBody(), "UTF-8"));  
 }  
 }, consumerTag -> {  
 });  
 String result = response.take();  
  
 channel.basicCancel(ctag);  
 return result;  
}

We establish a connection and channel.

Our call method makes the actual RPC request.

Here, we first generate a unique correlationId number and save it - our consumer callback will use this value to match the appropriate response.

Then, we create a dedicated exclusive queue for the reply and subscribe to it.

Next, we publish the request message, with two properties: replyTo and correlationId.

At this point we can sit back and wait until the proper response arrives.

Since our consumer delivery handling is happening in a separate thread, we're going to need something to suspend the main thread before the response arrives. Usage of BlockingQueue is one possible solutions to do so. Here we are creating ArrayBlockingQueue with capacity set to 1 as we need to wait for only one response.

The consumer is doing a very simple job, for every consumed response message it checks if the correlationId is the one we're looking for. If so, it puts the response to BlockingQueue.

At the same time main thread is waiting for response to take it from BlockingQueue.

Finally we return the response back to the user.

* And in server side:

DeliverCallback deliverCallback = (consumerTag, delivery) -> {  
 AMQP.BasicProperties replyProps = new AMQP.BasicProperties  
 .Builder()  
 .correlationId(delivery.getProperties().getCorrelationId())  
 .build();  
 String response = "";  
 try {  
 String message = new String(delivery.getBody(), "UTF-8");  
  
 int n = Integer.*parseInt*(message);  
  
 System.*out*.println(" [.] fib(" + message + ")");  
  
 response += *fib*(n);  
 } catch (RuntimeException e) {  
 System.*out*.println(" [.] " + e.toString());  
 } finally {  
 channel.basicPublish("", delivery.getProperties().getReplyTo(), replyProps,  
 response.getBytes("UTF-8"));  
 channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);  
 // RabbitMq consumer worker thread notifies the RPC server owner thread  
 synchronized (monitor) {  
 monitor.notify();  
 }  
 }  
};  
channel.basicConsume(*RPC\_QUEUE\_NAME*, false, deliverCallback, (consumerTag -> {  
}));

We use basicConsume to access the queue, where we provide a callback in the form of an object (DeliverCallback) that will do the work and send the response back.

**Question 3:**

Timeout: 60.0 seconds ...

Listing queues for vhost / ...

name messages\_ready messages\_unacknowledged

amq.gen-7rH7KU1PXH6hb-NxSTQmWg 0 0

amq.gen-wBIDPKnNatLn-xpeFWXlvQ 0 0

rpc\_queue 3 1

amq.gen-yZSFSJw3hgeOSygNAEgAuw 0 0

amq.gen-5leAe1ynrf7ufRAWe10m5g 0 0

The comment show list queues are running, with the number of message\_ready and number of messages\_unacknowledged.