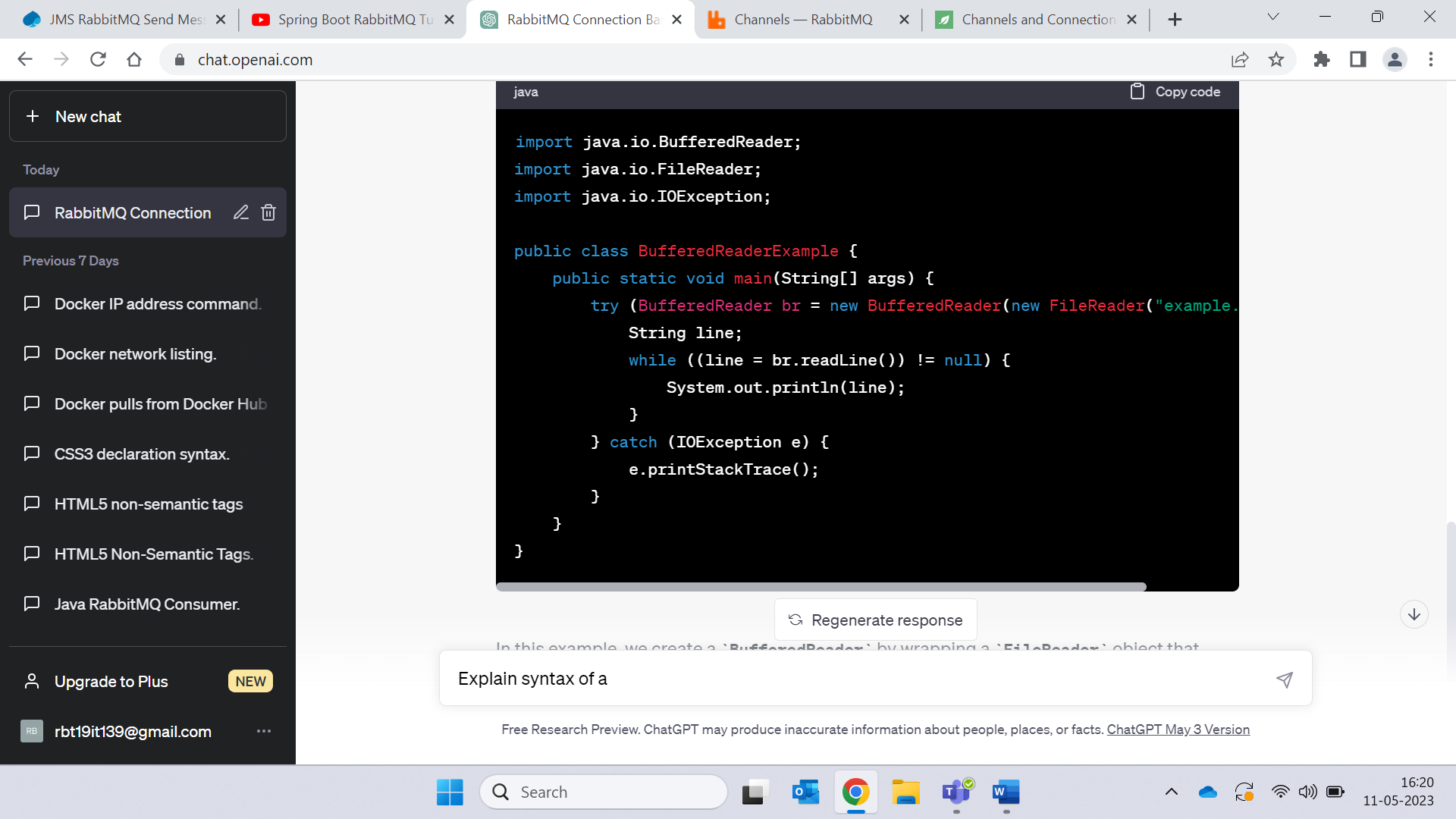
**<https://www.baeldung.com/java-rabbitmq-channels-connections>**

------------------------------------------------------------------------------------------------------------------------------------------Here's a simple example that demonstrates the usage of **BufferedReader** to read lines from a text file:

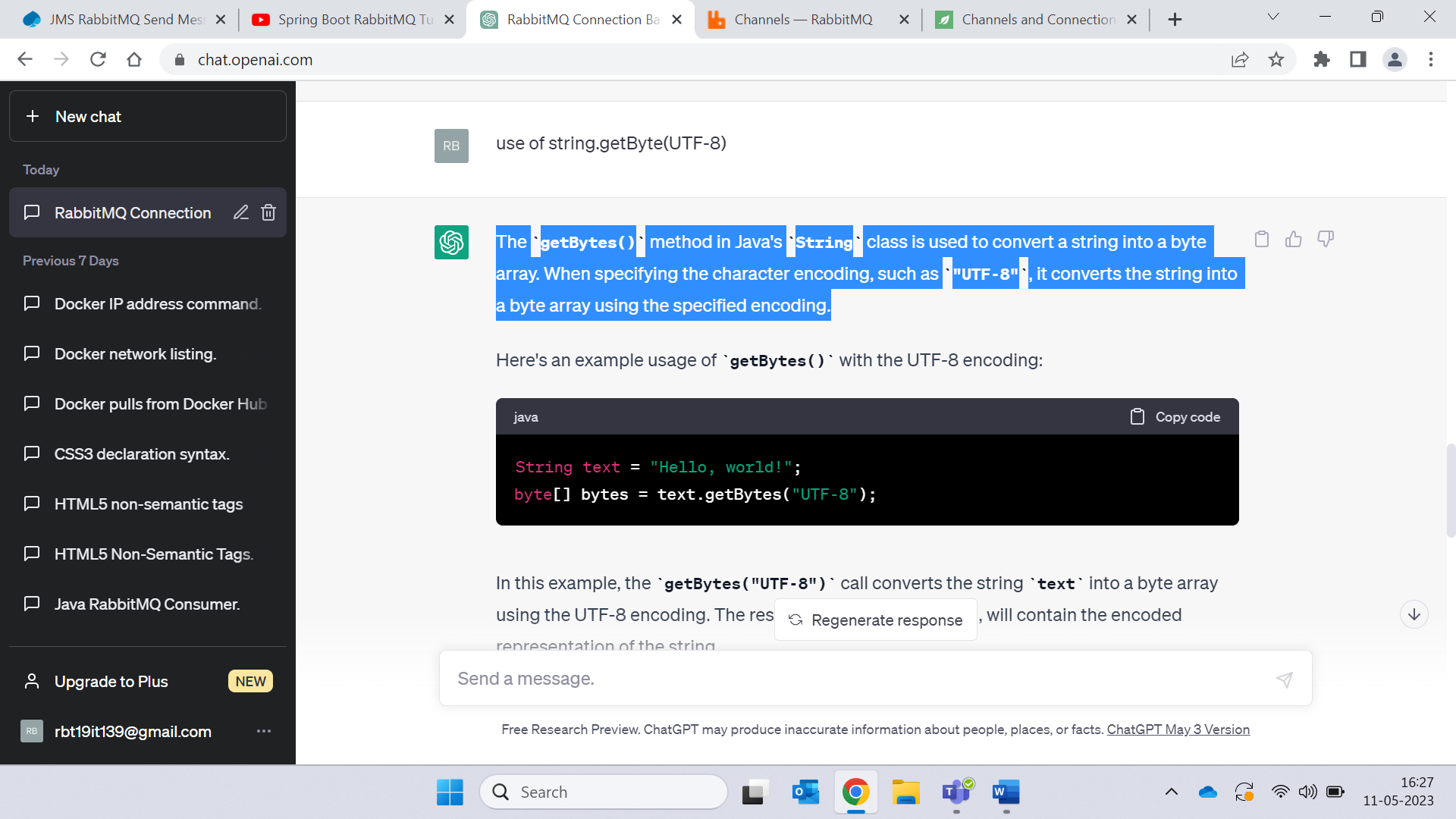
In this example, we create a **BufferedReader** by wrapping a **FileReader** object that reads from the "example.txt" file. We then use the **readLine()** method to read each line from the file and print it to the console. Finally, we handle any potential **IOException** that might occur during the reading process.

Remember to handle exceptions appropriately when working with **BufferedReader**, either by catching them or propagating them to the calling code.

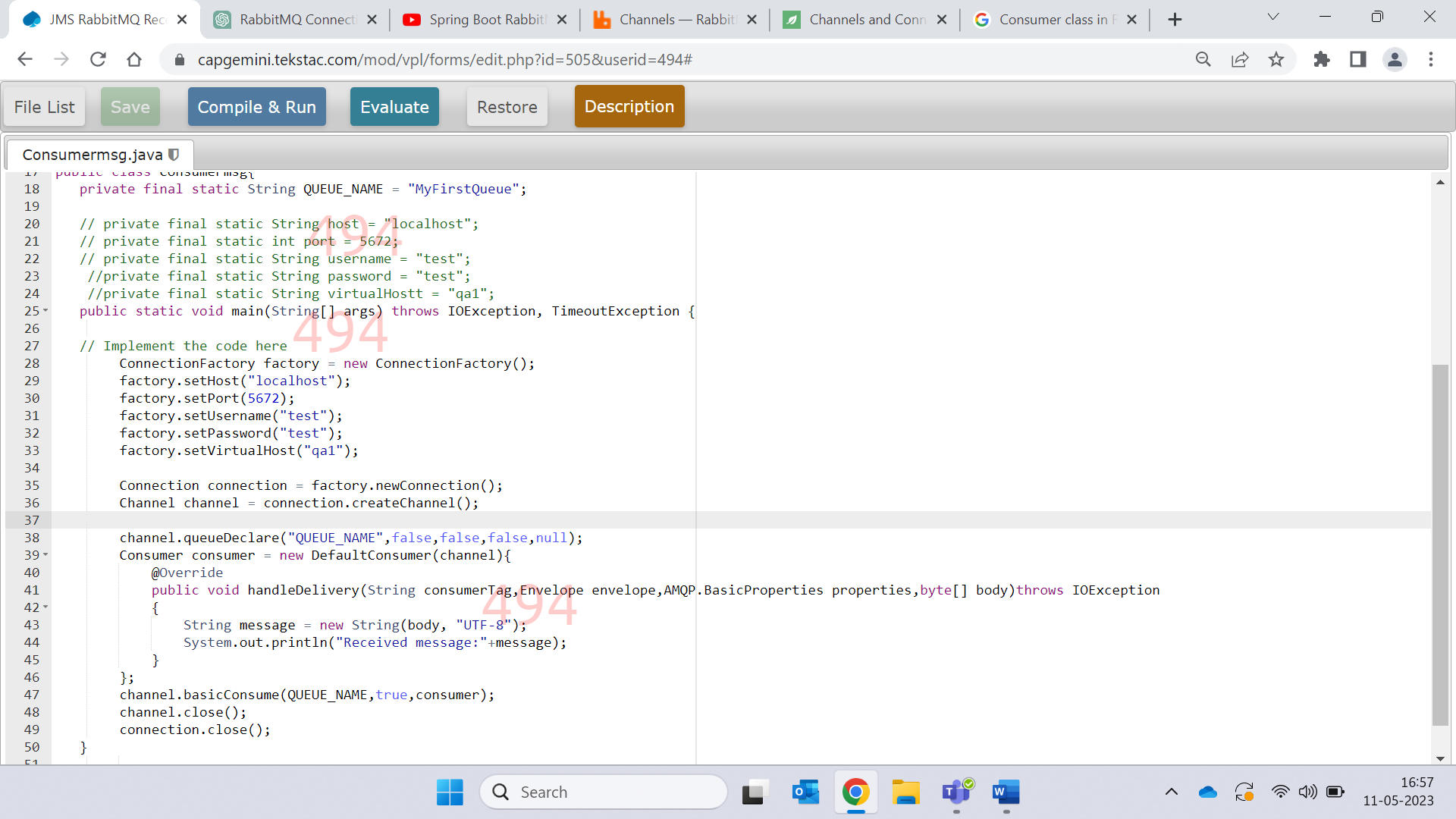
**UTF-8 🡪** In Java, UTF-8 (Unicode Transformation Format 8-bit) is a character encoding scheme that represents Unicode characters using variable-length encoding. It is widely used and supported in Java and many other programming languages as a standard for encoding text.

------------------------------------------------------------------------------------------------------------------------------------------

**getBytes()** 🡪 The **getBytes()** method in Java's **String** class is used to convert a string into a byte array. When specifying the character encoding, such as **"UTF-8"**, it converts the string into a byte array using the specified encoding.



------------------------------------------------------------------------------------------------------------------------------------------



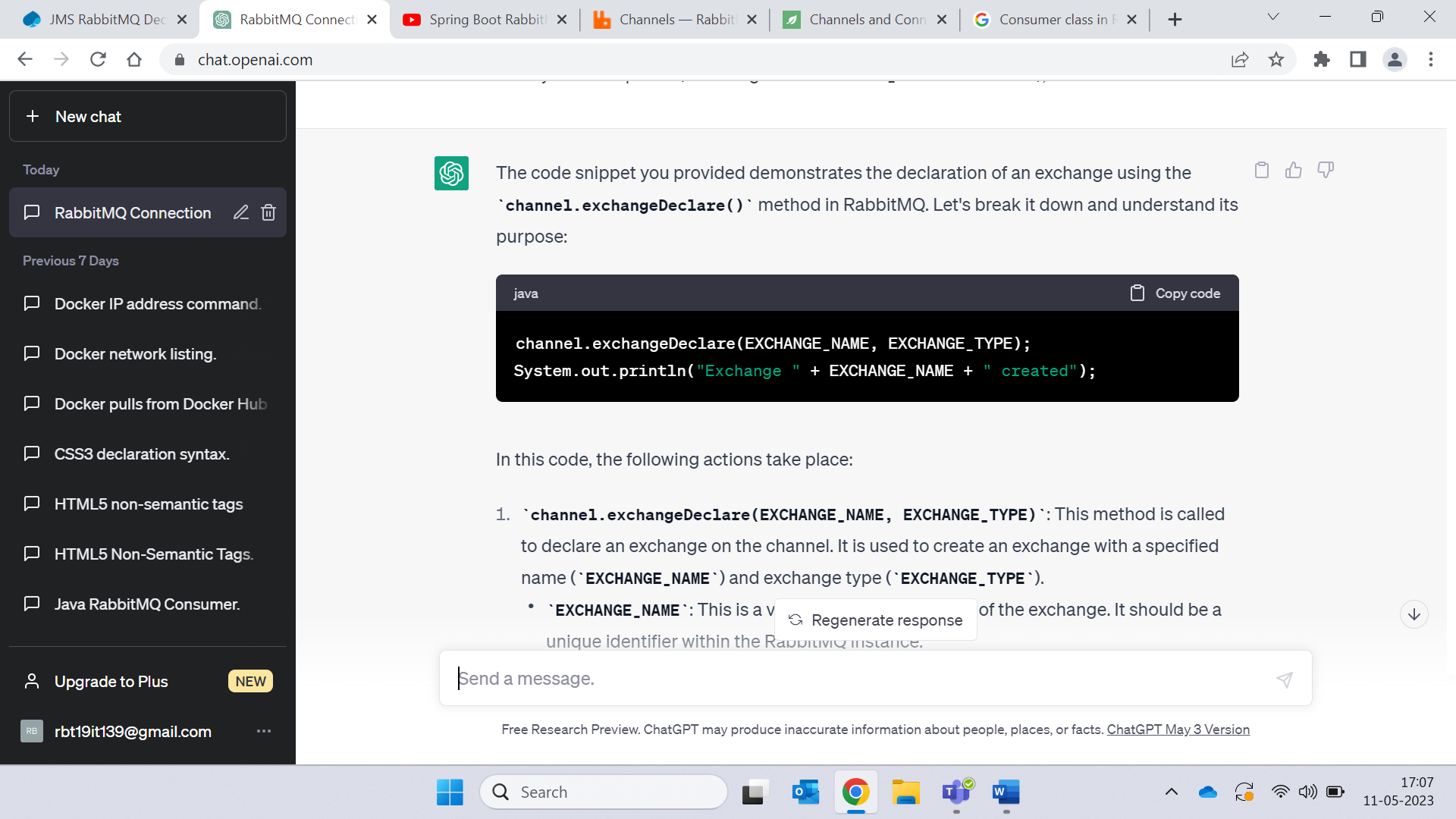
1. Create an instance of the **Consumer** class, implementing the **Consumer** interface or extending the **DefaultConsumer** class provided by the AMQP client library.
2. Implement the **handleDelivery()** method in your **Consumer** class. This method is called when a message is delivered to the consumer. You can define the logic to process the received message inside this method.
3. Register your consumer with the channel using the **basicConsume()** method. This tells RabbitMQ to start delivering messages to your consumer.

The **handleDelivery()** method you provided is a sample implementation of the **handleDelivery()** method in a **Consumer** class for RabbitMQ. This method is called by the RabbitMQ client library when a message is delivered to the consumer.

Let's break down the method and understand each parameter and its purpose:

* **consumerTag**: A unique identifier assigned by the server to identify the consumer. It is useful when you have multiple consumers listening to the same queue.
* **envelope**: An **Envelope** object containing metadata about the delivered message. It includes properties such as delivery tag, routing key, exchange name, and more.
* **properties**: An **AMQP.BasicProperties** object that holds the properties of the delivered message, such as content type, message headers, and delivery mode.
* **body**: A byte array that represents the message body. It contains the actual content of the delivered message.

In the provided implementation, the **handleDelivery()** method converts the **byte[]** message body into a **String** using the UTF-8 encoding and prints it to the console. It assumes that the message is encoded using UTF-8.



1. **channel.exchangeDeclare(EXCHANGE\_NAME, EXCHANGE\_TYPE)**: This method is called to declare an exchange on the channel. It is used to create an exchange with a specified name (**EXCHANGE\_NAME**) and exchange type (**EXCHANGE\_TYPE**).
   * **EXCHANGE\_NAME**: This is a variable holding the name of the exchange. It should be a unique identifier within the RabbitMQ instance.
   * **EXCHANGE\_TYPE**: This is a variable specifying the type of exchange to create, such as **"direct"**, **"topic"**, **"fanout"**, or **"headers"**. The exchange type determines the routing behavior for messages sent to the exchange.

The **exchangeDeclare()** method is responsible for creating the exchange with the specified name and type on the RabbitMQ server. If the exchange already exists, this method will verify that it matches the provided parameters. If the exchange does not exist, it will be created.

1. **System.out.println("Exchange " + EXCHANGE\_NAME + " created")**: This line is a simple output statement to indicate that the exchange was successfully created. It prints a message to the console, including the name of the created exchange.

After executing this code snippet, the exchange with the specified name and type will be declared on the channel, ready to be used for message routing and communication within RabbitMQ.

-----------------------------------------------------------------------------------------------------------------------------------------

**BuiltinExchangeType**

1. Direct Exchange: Routes messages to queues based on the message's routing key. Messages with a routing key that matches the binding key of a queue are routed to that queue.
2. Fanout Exchange: Broadcasts messages to all queues that are bound to it. It ignores the routing key of messages and delivers them to all connected queues.
3. Topic Exchange: Routes messages to queues based on pattern matching between the message's routing key and the bindings of the queues. The routing key can contain wildcard characters (**\*** and **#**) to match multiple routing keys.
4. Headers Exchange: Routes messages based on header attributes instead of the routing key. The headers exchange matches the headers in the message to the headers specified in the bindings of the queues.