



UNIVERSIDADE FEDERAL DE ITAJUBÁ  
INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO

COM242 - SISTEMAS DISTRIBUÍDOS  
PROF. RAFAEL FRINHANI

---

## TUTORIAL MQTT

*Message Queuing Telemetry Transport*

---

### Grupo:

BRUNO GUILHERME LUNARDI - nardi273@gmail.com - 2016003830  
CLAUDIO EDUARDO MACHADO SERPA - claudioserpa25@gmail.com - 2016010683  
LAIZA APARECIDA PAULINO DA SILVA - laizapaulino1@gmail.com - 2016001209  
THIAGO SILVA DA COSTA - thiagoscosta14@hotmail.com - 2016001826

03 de setembro, 2019

# Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>MQTT</b>	<b>2</b>
2.1	Publisher e Subscriber . . . . .	3
2.2	Definição de Broker . . . . .	3
2.3	Mensagem . . . . .	3
2.4	Payload . . . . .	4
2.5	QoS . . . . .	4
<b>3</b>	<b>Broker</b>	<b>4</b>
3.1	Mosca . . . . .	4
3.2	Emqttd . . . . .	5
3.3	Apache ActiveMQ and Apollo . . . . .	5
3.4	Mosquitto . . . . .	5
<b>4</b>	<b>Eclipse Paho java Client</b>	<b>7</b>
<b>5</b>	<b>Maven Project</b>	<b>8</b>
<b>6</b>	<b>Modelagem do sistema</b>	<b>8</b>
6.1	Diagrama de classe aplicação Envia_Calculo . . . . .	9
6.2	Diagrama de classe aplicação Envia_Resultado . . . . .	10
<b>7</b>	<b>Implementação do projeto Envia_Calculo em JAVA</b>	<b>10</b>
7.1	Criação do projeto . . . . .	10
7.2	Converter projeto para Maven Project . . . . .	11
7.3	Criando tela com o WindowBuilder Pro . . . . .	13
7.4	Implementação Classe Publisher - MQTT . . . . .	19
7.5	Teste do envio da mensagem da aplicação para o Mosquitto Broker . . . . .	23
<b>8</b>	<b>Implementação do projeto Envia_Resultado em JAVA</b>	<b>24</b>
8.1	Limpar cache do Mosquitto Broker . . . . .	27
8.2	Enviar resposta para o sistema Envia_Calculo . . . . .	27
8.3	Testando código . . . . .	28
<b>9</b>	<b>Implementação da mensagem de retorno no projeto Envia_Calculo</b>	<b>29</b>
<b>10</b>	<b>Criando arquivo de configuração para o IP do servidor</b>	<b>31</b>
<b>11</b>	<b>Teste Completo do sistema</b>	<b>31</b>
11.1	Limpando cache do Broker . . . . .	31
11.2	Descobrir IP do Broker no Linux . . . . .	32
11.3	Descobrir IP do Broker no Windows . . . . .	32
11.4	Estabelecendo comunicação entre as máquinas . . . . .	32
<b>12</b>	<b>Possíveis soluções para erro de comunicação</b>	<b>33</b>
<b>13</b>	<b>Código fonte das duas aplicações no Github</b>	<b>33</b>
<b>14</b>	<b>Conclusão</b>	<b>33</b>

---

## 1 Introdução

O crescimento da Internet das Coisas(IoT) e da presença de tecnologias como Machine-to-Machine(M2M) necessitam que os meios de conexão entre os mais diversos dispositivos acompanhem as necessidades destes e consigam evoluir de forma que permitam conexões mais rápidas, leves e eficientes. Neste âmbito, os sistemas distribuídos estão cada vez mais sofisticados e conectam cada vez mais uma variedade de hardwares distintos entre si em uma mesma rede, sejam eles roteadores, servidores, sensores sem fio, entre outros. Para tratar essa questão, diferentes protocolos de comunicação estão disponíveis no mercado, como o RMI, HTTP, MQTT(que foi usado neste trabalho) entre outros, cada um buscando suprir uma necessidade.

O MQTT é um protocolo de mensagens leves para sensores e pequenos dispositivos móveis. Ele utiliza o paradigma publish/subscribe para realizar a troca de mensagens. Neste paradigma é implementado um middleware chamado broker, que será o responsável por receber, enfileirar e enviar as mensagens recebidas dos publishers para os subscribers. O publisher se conecta ao broker para enviar a mensagem, já o subscriber se conecta ao brokers para receber a mensagem que tiver interesse.

Neste trabalho, foi implementado uma aplicação Java que realiza o processamento de uma operação matemática fazendo uso de um publicador e de um assinante. O cliente que requisita a operação seleciona os operadores, o tipo de operação e publica no tópico de interesse (publisher), essa mensagem é enviada para o "quebrador"(broker). O quebrador analisa o tópico e envia para o inscrito de interesse. O inscrito recebe a mensagem(subscriber), e a processa realizando a conta e então toma o papel publicador(publisher) e envia ao primeiro cliente que requisitou o processamento da conta(subscriber).

## 2 MQTT

MQTT, que é a sigla de Message Queue Telemetry Transport, é um protocolo de publicação e inscrição de mensagens leves, muito usado em aplicações de Internet das Coisas. Seu funcionamento ocorre de forma assíncrona desacoplando o emissor e o receptor no espaço e no tempo, dando a característica de maior escalabilidade [Yuan, 2017]. É leve e flexível devido ao uso do modelo publicação e assinatura que separa o publicador e o consumidor de dados.

Foi criado no fim de 1990, pela empresa de tecnologia IBM, com o foco em vincular sensores em pipelines de petróleo a satélites. Atualmente o protocolo é muito usado em aplicações de Internet das Coisas por ser altamente flexível e por apresentar um bom desempenho em redes de largura de banda limitada e alta latência. Um protocolo HTTP, embora esteja atrelado a internet, não poderia ser usado no lugar de um MQTT por diversas razões. O HTTP é síncrono, o que pode ser um problema, pois em "redes IOT" costuma se ter uma grande quantia de dispositivos conectados na rede, o que torna sua escalabilidade baixa. Sua transmissão funciona de um para um, e isso o torna custoso demais quando se tem um grande número de requisições.

No protocolo MQTT, tem-se um message broker e diversos clientes. O broker funciona recebendo mensagens e definindo para quais clientes deverá mandá-las. O cliente estabelece a comunicação com o broker assinando um tópico, então ele publica mensagens em um tópico que serão recebidas pelo broker, esse broker realiza o encaminhamento das mensagens para todos os clientes que fizeram a assinatura.

## 2.1 Publisher e Subscriber

Diferente do protocolo HTTP, que faz uso do modelo request/response, o protocolo MQTT faz uso do modelo publisher/subscribe. Esse modelo, também conhecido como publicador/assinante, faz com que quem deseje receber informações sobre algo precise inscrever-se.

Tanto o publisher quanto o subscriber são clientes. Um publicador embora realize troca de informações com um assinante, não o conhece. O publisher tem a responsabilidade de realizar a conexão com o broker e publicar mensagens. O subscriber se conecta ao servidor e recebe o que for de seu interesse [Team, 2015].

Esse protocolo faz uso de tópicos para poder filtrar as informações recebidas e para poder direcionar conteúdo para os assinantes. Especificamente, tópico é o nome dado ao assunto da mensagem, servindo para organizar as mensagens. A publicação de mensagens pode ocorrer tanto de um publicador para um ou de um publicador para vários assinantes de determinado tópico. Assim, é enviado apenas uma mensagem que é difundida entre todos os interessados e não um envio especificamente para cada inscrito do tópico. Isso se dá devido ao publicador e o assinante não estabelecerem uma conexão direta.

O protocolo MQTT possui um caráter não síncrono, assim consegue-se evitar a sobrecarga do broker e possíveis bloqueios até o cliente receber a mensagem. No entanto, a assincronia não é uma regra, havendo também a possibilidade de enviar mensagens síncronas.

A dissociação entre publisher e subscriber se dá pelo aspecto de espaço, tempo e sincronização. Quanto a espaço, não é necessário saber porta ou IP de nenhuma das partes. Quanto a tempo, não há necessidade de funcionarem ao mesmo tempo. Quanto a sincronização, as operações não precisam ser interrompidas no recebimento ou envio.

## 2.2 Definição de Broker

O broker é um servidor que é responsável por realizar a gestão das publicações e inscrições nos tópicos. Os principais são: Mosca, Emqttd, Apache ActiveMQ and Apollo, Mosquitto. Há diversas opções pagas e gratuitas, para esse trabalho e que será explicado em maior detalhes na seção X. Nesse trabalho o broker usado foi o Mosquitto, que é totalmente grátis.

O broker intermedia a comunicação entre os clientes publisher e subscriber, fazendo o recebimento e roteamento das mensagens para assinantes dos tópicos. O cliente ao iniciar uma conexão envia um pacote de conexão MQTT que contém seu username e senha, um identificador de cliente. O broker retorna uma mensagem CONNACK, que contém um indicador de sessão e um código de retorno que varia entre aceito e os erros que causaram uma conexão recusada.

## 2.3 Mensagem

Uma mensagem MQTT, possui uma matriz de bytes como carga útil (payload). Ela se torna persistente ao ser enviada com Qualidade de Serviço (QoS) 1 ou 2 [Team, 2019]. Seu formato básico consiste em:

- Cabeçalho fixo de 2 bytes
- Cabeçalho variável que pode conter tamanho de 0 a N bytes, dependendo da mensagem
- Payload que pode conter valor de 0 a N bytes.

Para receber uma mensagem, o cliente deve se conectar a um broker. Caso a conexão em uma sessão não persistente sofra alguma interrupção, os dados relativos ao tópico são perdidos e é necessário que se reconecte e inscreva-se novamente. Nesse protocolo, não se garante que o subscriber receberá a mensagem, apenas garante-se que o servidor broker a receberá [IBM, 2018a].

Uma mensagem pode ser definida como retida no broker. Quando isso ocorre, o sinalizador de retido recebe o valor true e ela fica guardada no broker junto do QoS do tópico. No momento em que um cliente realiza a inscrição em um tópico, ele automaticamente recebe essa mensagem. Ao se ter uma mensagem retida consegue-se eliminar a espera de se receber uma atualização de

um subscribe. Para se deletar uma mensagem retida deve-se enviar uma nova mensagem retida com carga útil com tamanho de zero byte, especificando o tópico.

## 2.4 Payload

O payload é a carga de dados relativa a uma transmissão, sendo definido como carga útil. No caso do MQTT, são excluídos dados como id do cliente, entre outros. Seus dados são binários e fazem correspondência ao tipo requerido na aplicação. Esse protocolo faz uso de carga útil para diminuir o tamanho da mensagem e garantir que as mensagens sejam o menor possível.

## 2.5 QoS

O QoS, Quality of Service, é a qualidade do serviço. O MQTT possui três qualidades, são elas: no máximo uma vez, no mínimo uma vez e exatamente uma vez. A qualidade é um atributo de `MqttMessage` [IBM, 2018b].

O QoS ao se configurar como no "máximo uma vez", tem o seu valor como 0. Isso significa que a mensagem não foi armazenada. Sua entrega ocorreu ou uma vez apenas ou não ocorreu, e sua entrega é desconhecida. Caso o cliente venha a se desconectar ou o servidor apresentar alguma falha, a mensagem será perdida. Embora não se tenha certeza sobre o recebimento ou envio da mensagem, esse modo de transferência é o mais rápido, também conhecido como fire and forget. Em caso de um cliente que esteja desconectado receber uma mensagem de QoS = 0, o MQTT poderá descartá-la conforme o broker usado.

Quando o QoS se configura como no "mínimo uma vez", o seu valor é 1. Esse é o modo de transferência padrão, que garante que a mensagem foi entregue já que ela deve ser entregue ao menos uma vez. O receptor pode recebê-la diversas vezes no caso de o emissor não receber uma confirmação, e será excluída após seu processamento.

Já quando o QoS se configura como "exatamente uma vez", o seu valor é 2. Isso significa que a mensagem foi entregue uma vez apenas, nem mais nem menos. Esse é o modo mais seguro, no entanto é o mais lento pois precisará de dois pares de transmissões entre emissor e receptor antes da mensagem ser excluída. No primeiro par de transmissões, o emissor manda a mensagem e espera o retorno, se não houver, ocorre a retransmissão. No segundo par, o emissor envia uma confirmação de que pode terminar o processamento.

## 3 Broker

O Broker é um intermediário que recebe os dados enviados para ele e se responsabiliza por encaminhá-los aos destinatários corretos. Dessa maneira pode-se desacoplar o produtor do cliente, sendo necessário se conhecer apenas o endereço do broker, possibilitando uma comunicação um-para-um, um-para-muitos e muitos-para-muitos[Andrei B. B. Torres, 2016].

### 3.1 Mosca

O Mosca é um broker MQTT Node.js que pode ser usado Standalone ou incorporado com outro aplicativo Node.js.

Compatível com o MQTT 3.1 e 3.1.1. Possui suporte apenas para QoS (Quality of Service) 0 e 1 e contém várias opções de armazenamento para pacotes offline de QoS 1[Mosca, 2016].

Para realizar a implementação do broker primeiramente se realiza a importação do modulo, depois se define a porta de operação do MQTT, por default será fornecida a porta padrão 1883, dentro do objeto Settings e o objeto é passado dentro da função Server para prepará-lo para o funcionamento. Por fim o broker é iniciado com o evento ready junto com uma função de retorno.

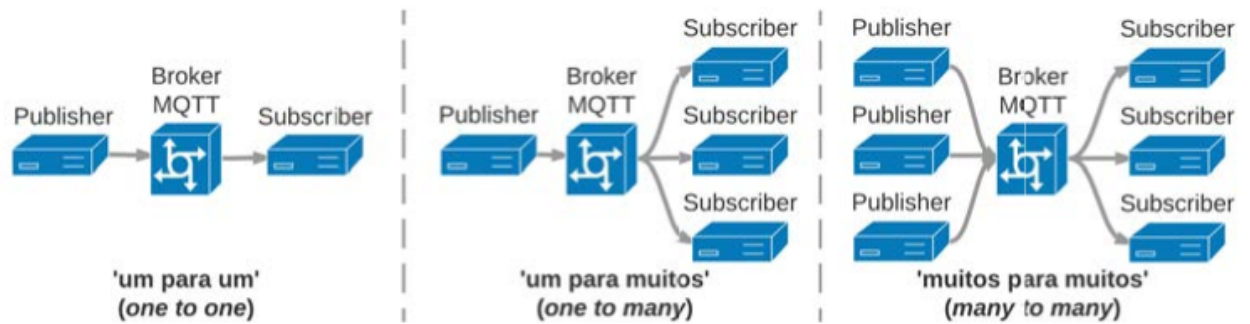


Figura 1: Tipos de distribuição de mensagem suportados pelo protocolo MQTT

```
var mosca = require('mosca');
var settings = {
  port: 1883
}

var server = new mosca.Server(settings);

server.on('ready', function() {
  console.log("ready");
});
```

Figura 2: Implementação Broker Mosca

### 3.2 Emqttd

É um broker MQTT que é distribuído, totalmente escalável e altamente disponível para aplicativos de IoT (Internet of Things), M2M (Machine to Machine) e Mobile.

Compatível com versões do MQTT V3.1 e V3.1.1, além de outros protocolos de comunicação como MQTT-SN, CoAP, LwM2M, WebSocket e STOMP. Suporta QoS 0,1 e 2. Escrito na linguagem Erlang.

Para instalá-lo no Windows basta realizar o download do pacote no site <https://emqttd-docs.readthedocs.io/en/latest/index.html>, descompactá-lo, abrir o prompt de comando, ir até a pasta bin do pacote e realizar o comando “emqttd console ” e depois o comando ”emqttd install”. Continuando no console pode se iniciar (“emqttd start”) ou parar (“emqttd stop”) o broker emqttd.

### 3.3 Apache ActiveMQ and Apollo

É um servidor de mensagens open source. Escrito em Java, mas suporta uma variedade de linguagens como C, C++, Ruby, Perl, Python, PHP. Suporta o MQTT V3.1. Pode-se realizar o download no site: <https://activemq.apache.org/components/classic/download/> Para ativar para o MQTT basta incluir um conector do broker usando a URL do MQTT.

### 3.4 Mosquitto

É um broker MQTT, leve e é adequado para uso em todos os dispositivos, desde computadores de placa única de baixa potência até servidores completos. O Mosquitto também fornece uma biblioteca C para implementar clientes MQTT e os comandos mosquitto pub e mosquitto sub

```
<transportConnectors>  
  <transportConnector name="mqtt" uri="mqtt://localhost:1883"/>  
</transportConnectors>
```

Figura 3: Código para ativar MQTT no Apache ActiveMQ and Apollo

[Mosquitto, 2019]. Mosquitto faz parte da Eclipse Foundation e suporta as versões 3.1.1 e 3.1 do MQTT.

Para realizar sua instalação, primeiro realize o download no site: <https://mosquitto.org/download/> de acordo com o computador e o sistema operacional. Neste caso será realizado no Windows 64 bits.

#### Windows

- [mosquitto-1.6.4-install-windows-x64.exe](#) (~1.4 MB) (64-bit build, Windows Vista and up, built with Visual Studio Community 2017)
- [mosquitto-1.6.4-install-windows-x32.exe](#) (~1.4 MB) (32-bit build, Windows Vista and up, built with Visual Studio Community 2017)

See also [readme-windows.txt](#) after installing.

Figura 4: Local de download do mosquitto

Após o download, se deve executar o executável que foi baixado. Será iniciado o processo de instalação, na tela de apresentação selecione o next.

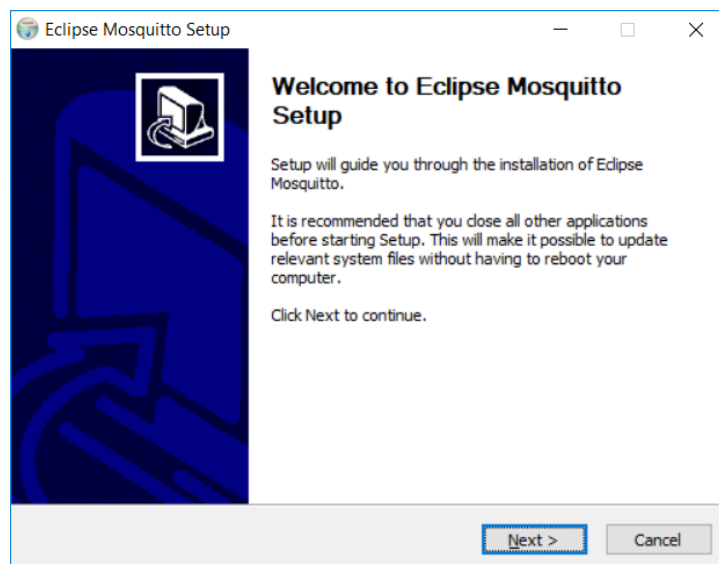


Figura 5: Tela de apresentação da instalação do mosquitto

Depois se selecionam os componentes a serem instalados, por default todos vem marcados, deixe assim e selecione o next.

Após isso escolha o local onde o mosquito será salvo e clique em install.

Após encerrar a instalação aparecerá a tela de conclusão e clique e Finish.

Por fim abra o Prompt de Comando, navegue até a pasta onde está instalado o mosquitto e execute o comando “net start mosquitto” para iniciar o mosquitto broker.

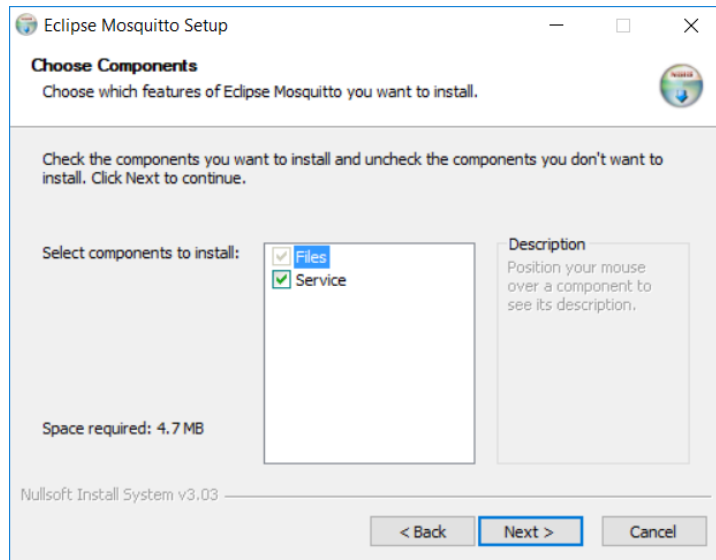


Figura 6: Tela de seleção de componentes do mosquitto

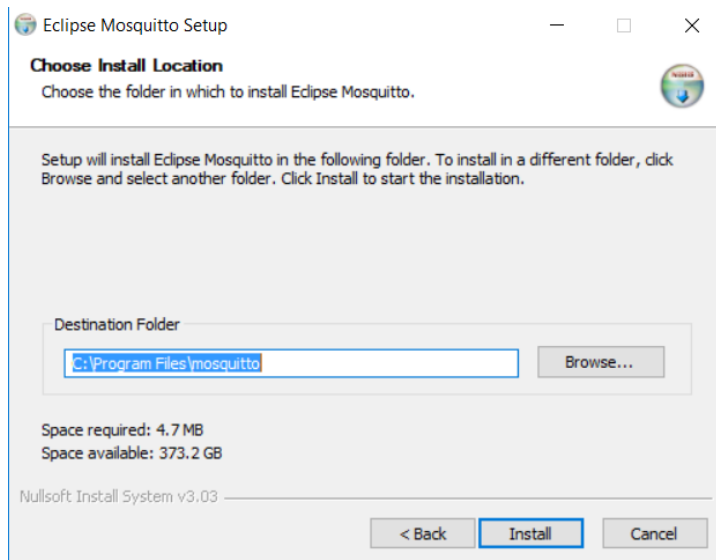


Figura 7: Tela de escolha do local onde será salvo o mosquitto

## 4 Eclipse Paho java Client

O Eclipse Paho é um projeto com implementações de clientes MQTT e MQTT-SN em diferentes linguagens de programação. Foi uma das primeiras implementações de cliente MQTT de código aberto disponíveis e é mantida pela sua comunidade. A versão Java do Eclipse Paho é usada para conectar-se aos brokers MQTT.

A API síncrona do Paho facilita a implementação da lógica MQTT dos aplicativos. Enquanto a API assíncrona fornece ao desenvolvedor um controle total do MQTT permitindo um alto desempenho. O Paho suporta todos os recursos do MQTT e uma comunicação segura com o MQTT Broker via TLS.

A maneira mais conveniente de instalar o Paho é usar uma ferramenta de gerenciamento de dependências como Maven e será mais detalhada no capítulo 7.



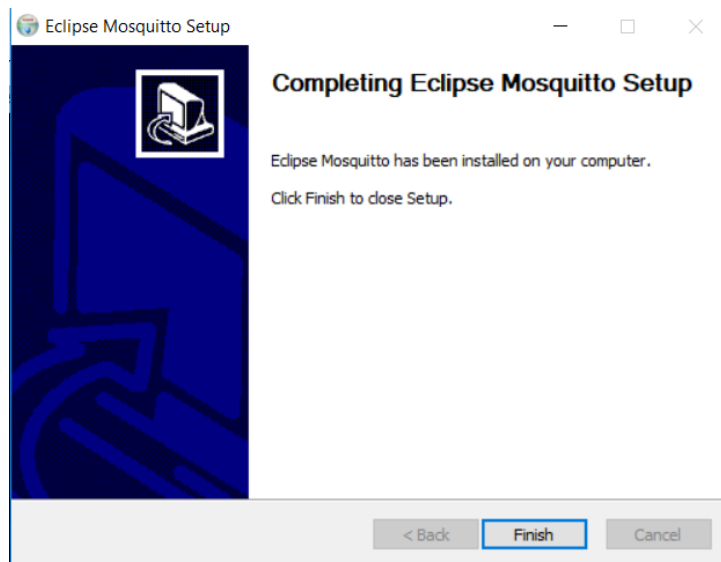


Figura 8: Tela de conclusão de instalação do mosquito

```
C:\Program Files\mosquitto>net start mosquitto
O serviço de Mosquitto Broker está sendo iniciado.
O serviço de Mosquitto Broker foi iniciado com êxito.
```

Figura 9: Comando para iniciar o mosquitto broker

## 5 Maven Project

Maven é uma ferramenta usada para criar e gerenciar qualquer projeto baseado em Java, que facilita o desenvolvimento e compreensão dos projetos[Maven, 2019]. O principal objetivo do Maven é permitir ao desenvolvedor conhecer o estado completo do desenvolvimento no menor período de tempo, para isso o Maven procura facilitar o processo de criação; fornecer um sistema de construção uniforme; fornecer informações de qualidade do projeto; fornecer diretrizes para o desenvolvimento de melhores praticas e permitir a migração de novos recursos. O Maven é fornecido no seguinte link: <https://maven.apache.org/download.cgi>.

## 6 Modelagem do sistema

O sistema que exemplificará o uso do Mqtt é composto por duas aplicações. Uma é responsável por enviar dois valores e um operador para poder realizar cálculos (aplicação Envia\_Calculo). O outro enviará o resultado da operação (aplicação Envia\_Resultado). E a comunicação destes acontecerá através de um broker. Este funcionamento é esquematizado na Figura 10

De acordo com a Figura 10, é possível notar que para a execução deste sistema foi criado dois tópicos. O tópico "enviar\_conta" é publicado ao broker pela aplicação Envia\_Calculo e conterá uma mensagem com os dois valores numéricos e o operador para realização de cálculos. O sistema Envia\_Resultado estará inscrito no tópico "enviar\_conta", para receber do broker a mensagem, separar os valores e o operador e realizar o devido cálculo.

O tópico "enviar\_resultado" é publicado ao broker pela aplicação Envia\_Resultado, na qual enviará o resultado da operação realizada para o broker. A aplicação Envia\_Calculo estará

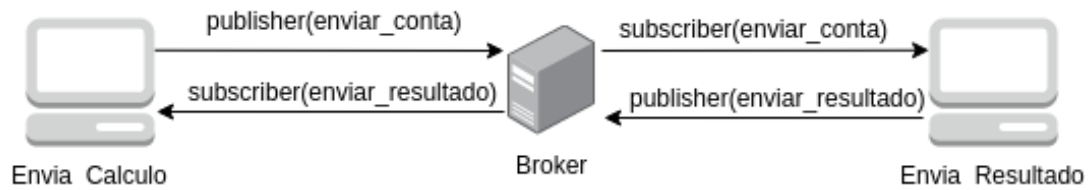


Figura 10: Modelagem geral do sistema

inscrita no tópico "enviar\_resultado" e receberá do broker a mensagem contendo o resultado.

A seguir é apresentado os diagramas de classes das aplicações Enviar\_Calculo e Enviar\_Resultado.

## 6.1 Diagrama de classe aplicação Enviar\_Calculo

Esta aplicação foi desenvolvida com base na modelagem da Figura 11

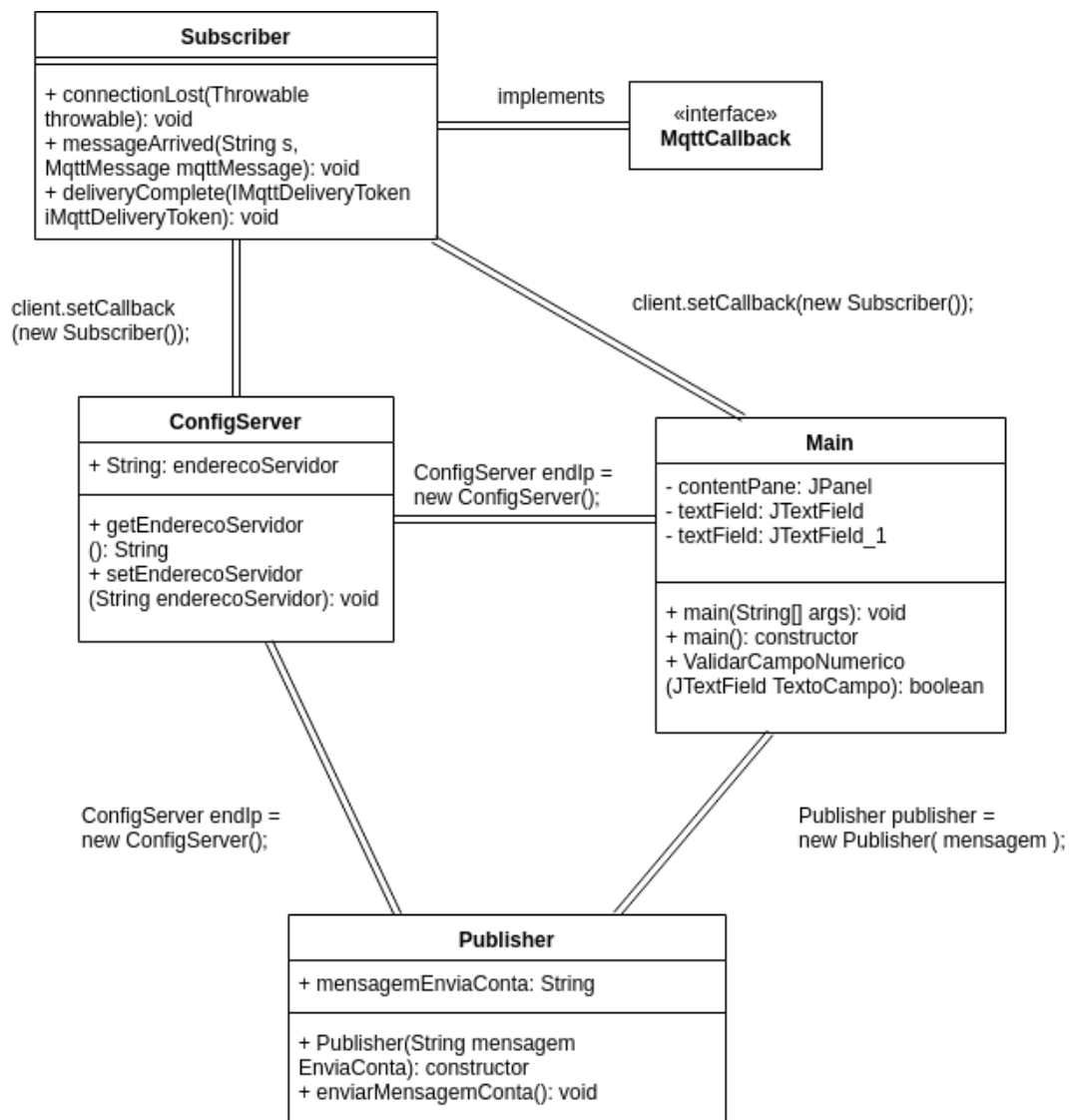


Figura 11: Diagrama de classe da aplicação Enviar\_Calculo

A classe Main é a responsável por iniciar a aplicação. A Publisher enviará os valores e

operadores para o broker, pelo tópic "enviar\_conta". A classe ConfigServer é a responsável por conter o endereço do servidor broker e a Subscriber é a classe que estará inscrita no tópico "enviar\_resultado", ou seja, responsável por receber o resultado da operação.

## 6.2 Diagrama de classe aplicação Envia\_Resultado

Esta aplicação foi desenvolvida com base na modelagem da Figura 12

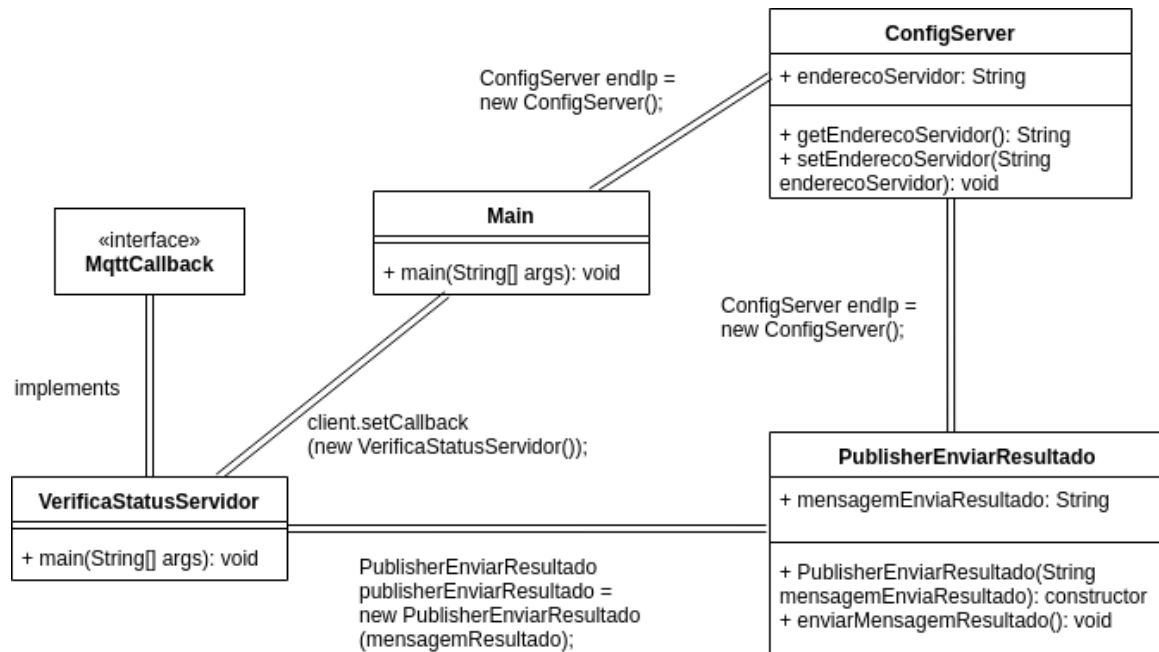


Figura 12: Diagrama de classe da aplicação Envia\_Resultado

A classe Main inicializa a aplicação. A ConfigServer contém o IP do broker. A PublisherEnviarResultado envia a mensagem com o resultado da operação para o broker, por meio do tópico "enviar\_resultado". A classe VerificaStatusServidor é a que recebe mensagem do broker, pelo tópico "enviar\_conta".

## 7 Implementação do projeto Envia\_Calculo em JAVA

A seguir será apresentado a forma como foi elaborado o sistema que enviará dois números e um operador para outro sistema, que tratará a conta destes e enviará o resultado de volta.

### 7.1 Criação do projeto

Abrir o eclipse, selecione o workspace de sua preferência e depois clique em "Launch", conforme Figura 13

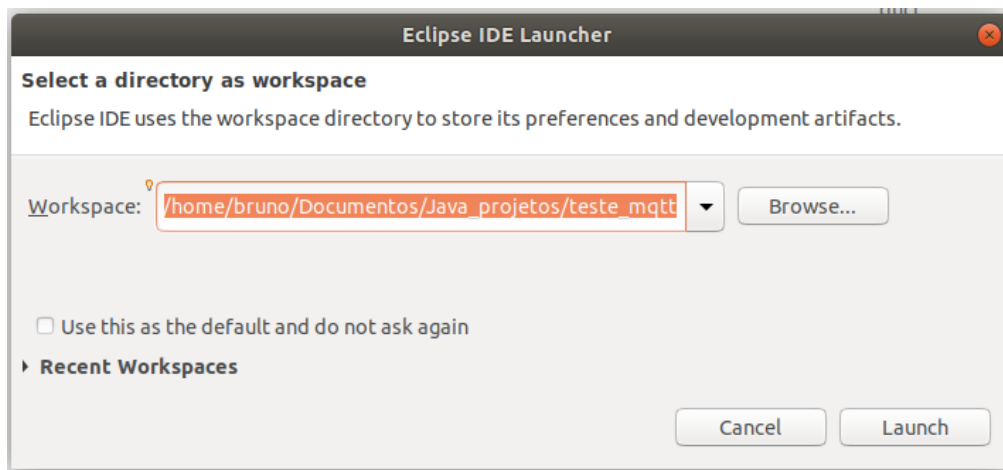


Figura 13: Tela de Workspace do Eclipse

Em seguida clique em File - New - Java Project. Na tela que abrir coloque o nome do projeto, conforme exemplo da Figura 14

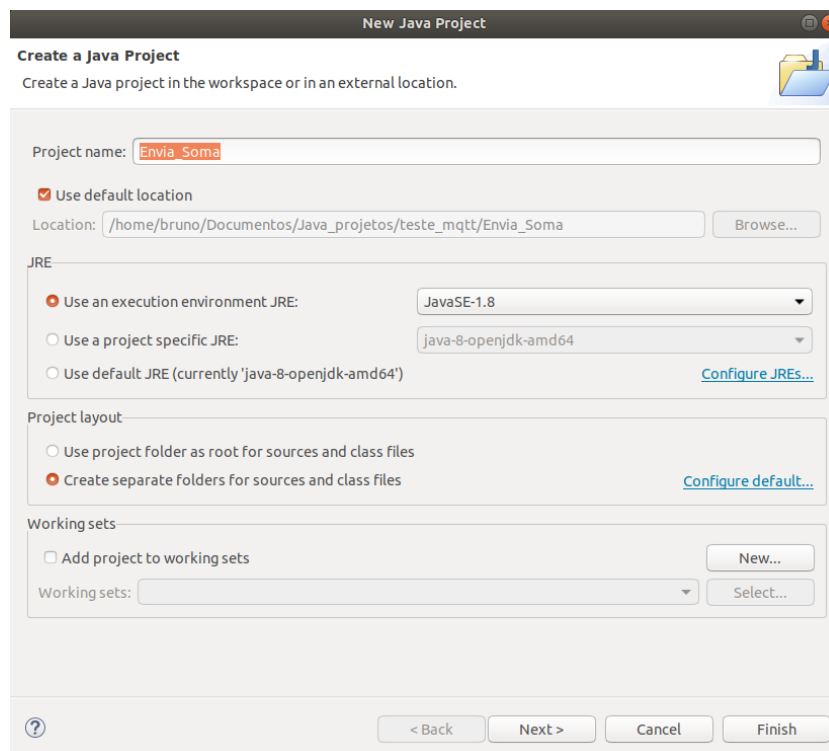


Figura 14: Tela de criar projetos do Eclipse

Após nomear o projeto clique em Finish.

## 7.2 Converter projeto para Maven Project

Para facilitar a adição da biblioteca Paho é necessário converter o projeto para Maven, para isto basta clicar com o botão direito do mouse sobre o projeto - Configure - Convert to Maven Project. Abrirá a janela com o título Create new POM, clique em Finish, conforme Figura 15

**Create new POM**

**Maven POM**

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /Envia\_Soma

Artifact:

Group Id: Envia\_Soma

Artifact Id: Envia\_Soma

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Cancel Finish

Figura 15: Tela de converter projeto em Maven project

Após converter o projeto para Maven, o próximo passo é adicionar a biblioteca Paho, para utilizar o protocolo MQTT. Para isto, basta adicionar a seguinte dependência no arquivo pom.xml do projeto, conforme Figura 16



```

1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocat
2 <modelVersion>4.0.0</modelVersion>
3 <groupId>Envia_Soma</groupId>
4 <artifactId>Envia_Soma</artifactId>
5 <version>0.0.1-SNAPSHOT</version>
6
7 <!-- Dependencia do MQTT -->
8 <dependencies>
9 <!-- https://mvnrepository.com/artifact/org.eclipse.paho/org.eclipse.paho.client.mqttv3 -->
10 <dependency>
11 <groupId>org.eclipse.paho</groupId>
12 <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
13 <version>1.1.0</version>
14 </dependency>
15
16 </dependencies>
17
18 <build>
19 <sourceDirectory>src</sourceDirectory>
20 <plugins>
21 <plugin>
22 <artifactId>maven-compiler-plugin</artifactId>
23 <version>3.8.0</version>
24 <configuration>
25 <source>1.8</source>
26 <target>1.8</target>
27 </configuration>

```

Figura 16: Dependências do Paho adicionada via Maven

### 7.3 Criando tela com o WindowBuilder Pro

Para a construção das telas do programa em desenvolvimento será utilizado o WindowBuilder Pro, que auxilia no desenvolvimento de telas para o Eclipse no padrão Drag and Drop.

Clique no seguinte link. Na tabela Update Sites, clique com o botão direito do mouse sobre a coluna da tabela "Zipped Update Site" e copie a URL da versão correspondente ao seu Eclipse. Tabela apresentada na Figura 17. Para o nosso exemplo o link copiado foi:

org.eclipse.wb.releng.update.site - <http://download.eclipse.org/windowbuilder/latest/>

## Installing WindowBuilder Pro

All downloads are provided under the terms and conditions of the [Eclipse Foundation Software User Agreement](#) unless otherwise specified.

Develop Java graphical user interfaces in minutes for Swing, SWT, RCP and XWT with WindowBuilder Pro's WYSIWYG, drag-and-drop interface. Use wizards, editors and intelligent layout assist to automatically generate clean Java code, with the visual design and source always in sync.

These instructions assume that you have already installed some flavor of Eclipse. If you have not, Eclipse can be downloaded from <http://www.eclipse.org/downloads/>. Instructions and system requirements for installing WindowBuilder can be found [here](#).

### Update Sites

	Download	
Version	Update Site	Zipped Update Site
Latest (1.9.2)	<a href="#">link</a>	<a href="#">link</a>
Last Good Build	<a href="#">link</a>	<a href="#">link</a>
Gerrit	<a href="#">link</a>	<a href="#">link</a>
1.9.2 (Permanent)	<a href="#">link</a>	<a href="#">link</a>
1.9.1 (Permanent)	<a href="#">link</a>	<a href="#">link</a>
1.9.0 (Permanent)	<a href="#">link</a>	<a href="#">link</a>

Figura 17: WindowBuilder Pro link para instalação

Em seguida, no Eclipse, clique em Help - Install New Software e na janela "Available Software" cole a URL que você copiou do passo anterior e aperte Enter. Na guia Name selecione todos que aparecerem e clique em Next, conforme Figura 18.

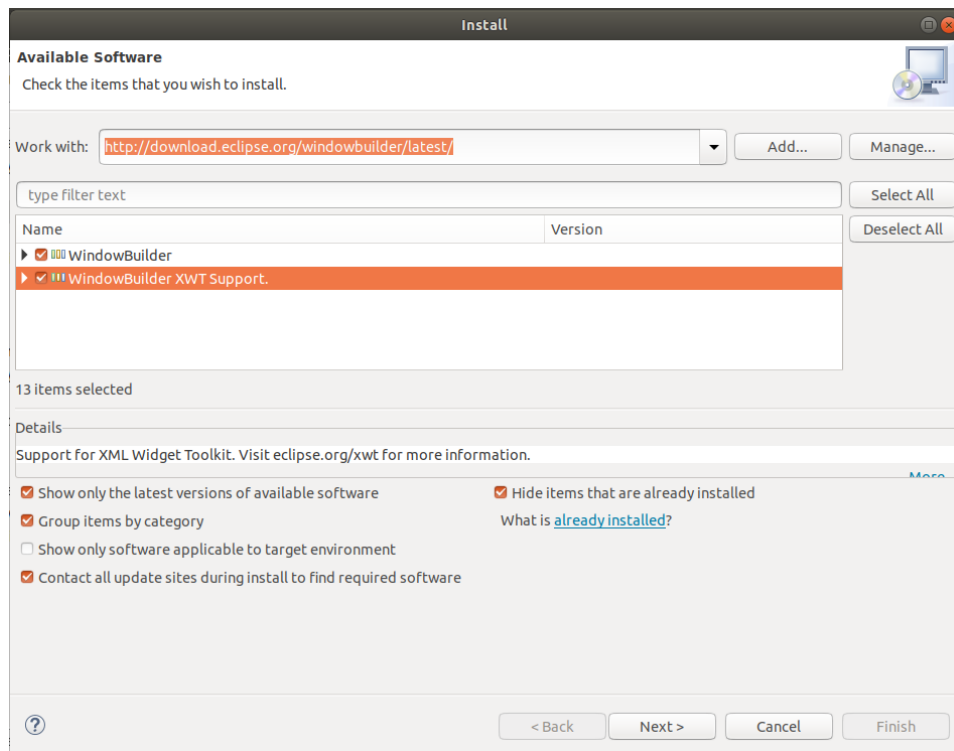


Figura 18: Instalação WindowBuilder Pro

Em seguida clique em Next novamente, aceite todos os temas e por fim clique em Finish e aguarde o fim da instalação e reinicie o Eclipse.

Após o Eclipse inicializar clique com o botão direito do mouse sobre o pacote do projeto (caso você tenha criado um pacote - para fazer isto basta clicar com o botão direito do mouse sobre a pasta src do seu projeto, depois clicar em new e por fim Package e nomeá-lo) - New - Other, na janela "Select a wizard", clique em WindowBuilder - Swing Designer - JFrame. Após estes passos clique em "Next", conforme Figura 19.

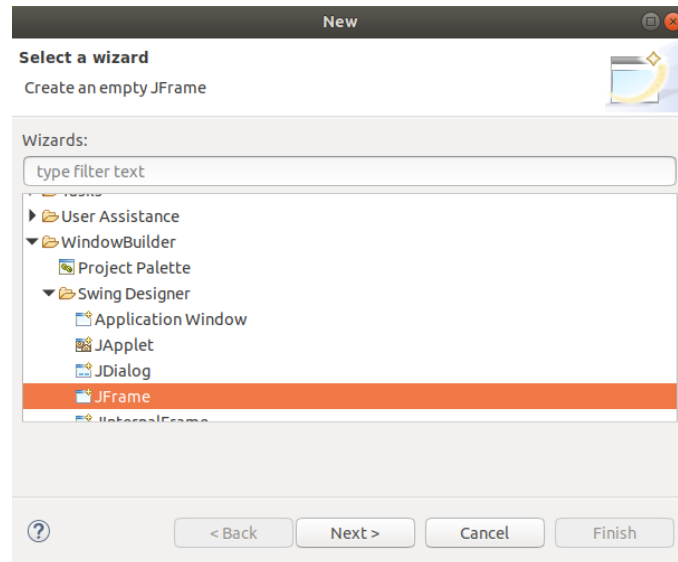


Figura 19: Selecionar JFrame WindowBuilder Pro

Na janela "New JFrame" escolha um nome para a tela, por exemplo Main, conforme Figura 20

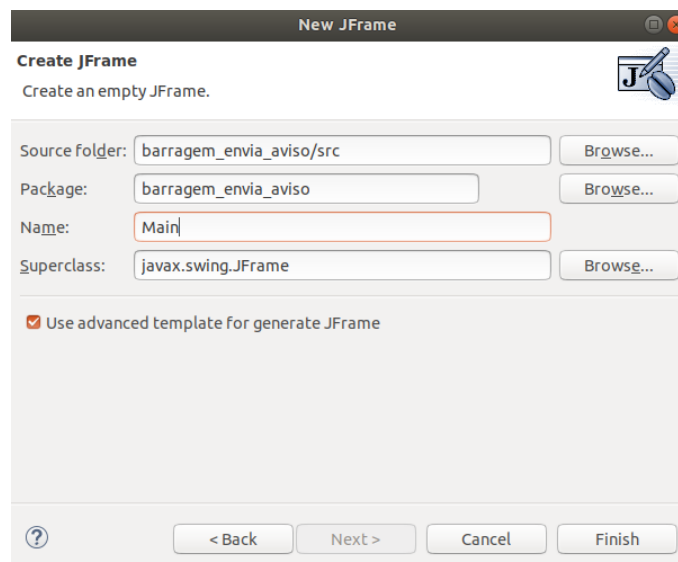


Figura 20: Nomear JFrame WindowBuilder Pro



Clique em "Finish".

No Eclipse, abra o JFrame que foi criado e na parte de baixo dos códigos clique em "Design", igual a Figura 21

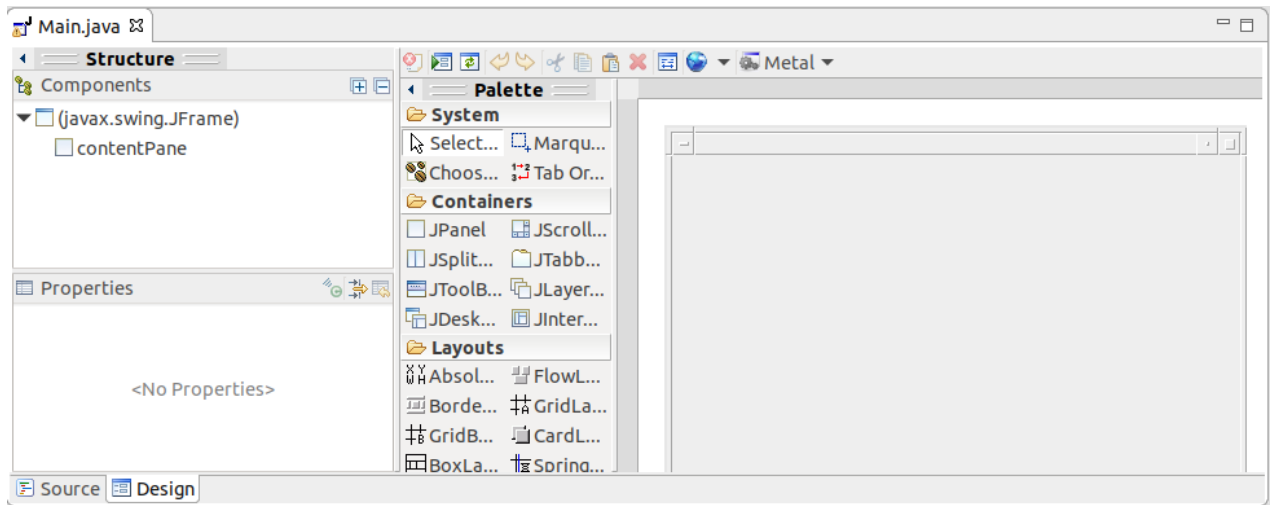


Figura 21: Design JFrame WindowBuilder Pro

Na janela que foi aberta, Na aba "Palette" procure por "Layouts" e em seguida clique no layout "Absolute Layout" e depois clique no painel (ambos com o botão esquerdo do mouse). Este layout permite posicionar os componentes visuais livremente na tela.

Em seguida clique em JLabel e depois clique no painel, vamos repetir este passo 3 vezes (para adicionar 3 JLabel na tela proposta). Agora realize o mesmo passo, só que adicione dois JTextField em vez de um JLabel e depois faça a mesma coisa para adicionar um JComboBox e um JButton. Agora posicione e redimensione os objetos visuais, para que a sua tela fique semelhante a da Figura 22



Figura 22: Início Criação de tela

Agora esta tela será modificada seus parâmetros. Clique em "Source", na aba ao lado de "Design", onde ficam os códigos do programa. Altera o título das JLabels da seguinte maneira:

```

JLabel lblNewLabel = new JLabel("Valor 1:");
JLabel lblNewLabel = new JLabel("Operador:");
JLabel lblNewLabel = new JLabel("Valor 2:");

```

A JComboBox conterá os operadores de contas mais simples, como soma, divisão multiplicação e subtração. Para isto, altere o código da JComboBox da seguinte maneira, igual a Figura 23:

```

JComboBox comboBox = new JComboBox();
comboBox.setBounds(162, 66, 111, 24);
comboBox.addItem("+");
comboBox.addItem("-");
comboBox.addItem("/");
comboBox.addItem("*");
contentPane.add(comboBox);

```

Figura 23: Código para adicionar os operadores na JComboBox

Por fim será alterado o label do botão, localize a linha de código:

```

JButton btnNewButton = new JButton("New button");

```

E altere para

```

JButton btnNewButton = new JButton("Enviar Conta");

```

A tela deve ficar semelhante a apresentada na Figura 24



Figura 24: Tela inicial projeto Enviar Conta

Para realizar um teste nesta tela da aplicação, altera o código do JButton da seguinte maneira, apresentada na Figura 25:

```

JButton btnNewButton = new JButton("Enviar Calculo");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {

        System.out.println(textField.getText() + " " +
            comboBox.getSelectedIndex() + " " +
            textField_1.getText());
    }
});
btnNewButton.setBounds(159, 175, 142, 25);
contentPane.add(btnNewButton);

```

Figura 25: Código para testar JButton

Agora para iniciar esta aplicação clique com o botão direito do mouse no projeto e depois em Run As e em seguida Java Application. Na Janela "Save and Launch" verifique se Main.java está selecionado, se sim clique em Ok. Este teste apenas exibirá a tela e imprimirá no console qual o valor que foi digitado nos dois campos de texto, após clicar no botão. O texto será impresso no console do Eclipse, conforme exemplo da Figura 26:

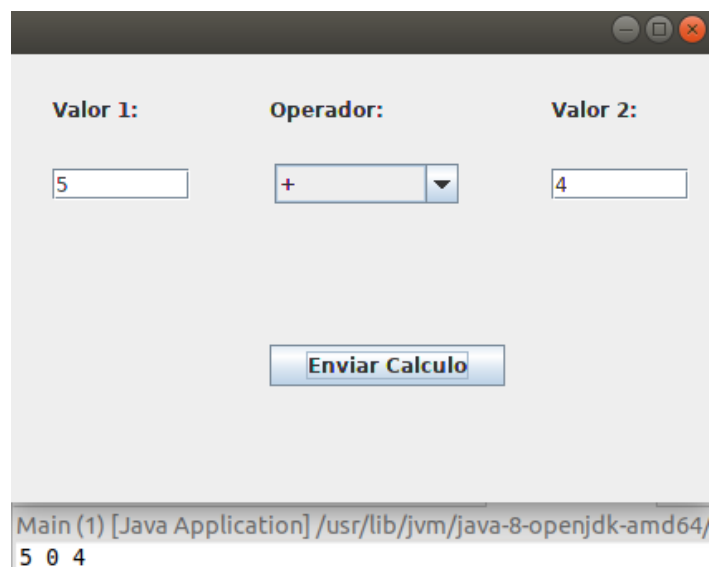


Figura 26: Teste do JButton da tela inicial

Agora é preciso garantir que os campos Valor 1 e Valor 2 sejam apenas número. para isto será adicionado o seguinte método, após o construtor public Main(), de acordo com a Figura 27.

Agora para tratar os problemas de import, no começo do código digite a seguinte importação, igual a Figura 28.

```
//Método para validar JTextField como número
public boolean ValidarCampoNumerico(JTextField TextoCampo) {
    long valor;
    if (TextoCampo.getText().length() != 0){
        try {
            valor = Long.parseLong(TextoCampo.getText());
        }catch(NumberFormatException ex){
            JOptionPane.showMessageDialog(null, "Digite um valor válido!", "Erro de valor",
                JOptionPane.INFORMATION_MESSAGE);
            TextoCampo.grabFocus();
            return false;
        }
    }else {
        JOptionPane.showMessageDialog(null, "Digite um número", "Erro de valor",
            JOptionPane.INFORMATION_MESSAGE);
        TextoCampo.grabFocus();
        return false;
    }
    return true;
}
```

Figura 27: Método para validar campo numérico

```
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
```

Figura 28: Importação do JOptionPane

Para fazer um novo teste, basta adicionar a chamada do método no evento de clique de botão, igual na Figura 29:

```
JButton btnNewButton = new JButton("Enviar Soma");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        if( ValidarCampoNumerico(textField) && ValidarCampoNumerico(textField_1) ) {

            System.out.println(textField.getText() + " " +
                comboBox.getSelectedIndex() + " " +
                textField_1.getText());

        }
    }
});
btnNewButton.setBounds(159, 175, 142, 25);
contentPane.add(btnNewButton);
}
```

Figura 29: Chamada do método de validar campo

## 7.4 Implementação Classe Publisher - MQTT

Esta classe será responsável por criar um tópico para enviar mensagens ao Broker Mosquitto Eclipse. Para isto, primeiramente crie uma nova classe Java, clique com o botão direito do mouse

no pacote do seu projeto (src/nome\_pacote) - New - Class. Na janela "New Java Class" nomeie a nova classe, de acordo com a Figura 30 e depois clique em Finish.

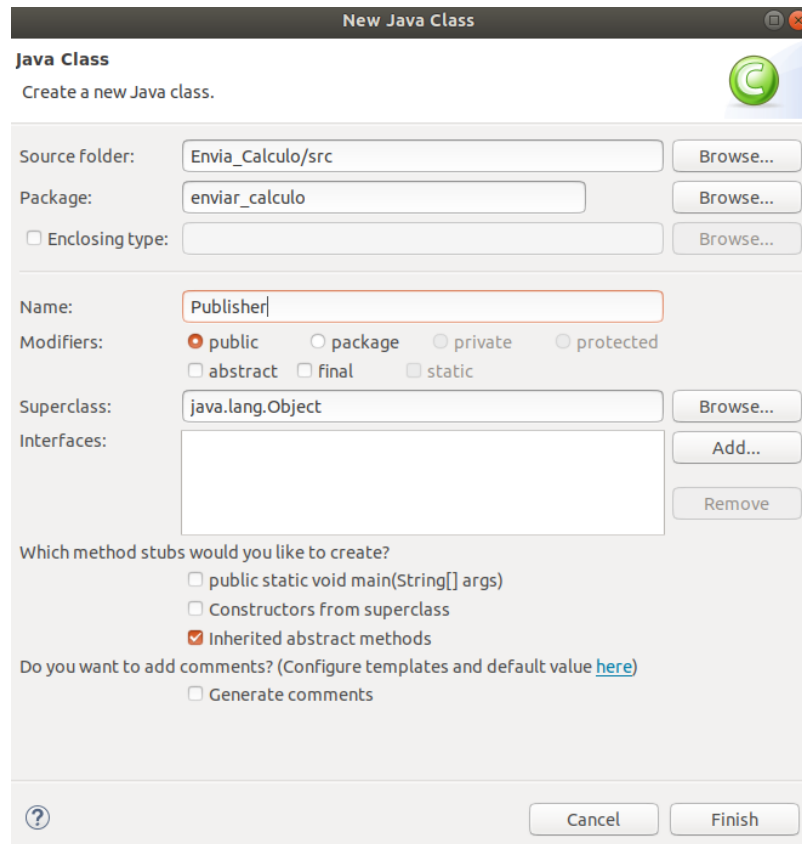


Figura 30: Nova Classe Java

Após a criação da classe agora deve-se adicionar os seguintes imports para trabalhar com o protocolo MQTT, igual a Figura 31

```
package enviar_calculo;

//Imports para utilizar o protocolo MQTT
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
```

Figura 31: Importação para MQTT

Agora será definido o construtor da classe, que receberá a mensagem que será enviada a outro sistema (valor1 + operador + valor2), de acordo com a Figura ??

```
//String que enviará a conta para o outro sistema
String mensagemEnviaConta;
//construtor da classe para enviar a mensagem
public Publisher(String mensagemStatusBarragem){
    this.mensagemEnviaConta = mensagemStatusBarragem;
}
```

Figura 32: Construtor da classe Publisher

A seguir é exibido, com comentários, o código que enviará a mensagem ao broker, com o tópico enviar\_conta. Figura 33:

```
//Método para enviar a mensagem para o mosquitto
public void enviarMensagemConta() throws MqttException {
    //Local para onde será enviado a mensagem (broker)
    MqttClient client = new MqttClient("tcp://localhost:1883", MqttClient.generateClientId());
    //abre conexão com o broker (Mosquitto)
    client.connect();
    //objeto de envio de mensagem do broker
    MqttMessage mensagem = new MqttMessage();
    //Payload, conteúdo da mensagem montagem da mensagem que será enviada ao broker
    mensagem.setPayload(mensagemEnviaConta.getBytes());
    //Publica a mensagem, com seu tópico (para alguém se escrever neste tópico) e a mensagem mont
    client.publish("enviar_conta", mensagem);
    //fecha conexão com broker mosquitto
    client.disconnect();
}
```

Figura 33: Construtor da classe Publisher

A classe MqttClient é a responsável por comunicar um sistema a um servidor broker. Em seu construtor é definido o endereço do broker. Após definir o endereço do servidor o próximo passo é abrir a conexão com o broker, através do método connect().

A classe MqttMessage define as especificações da mensagem que será enviada ao broker. O método setPayload() define o tamanho do conteúdo da mensagem. O método publish(), da classe MqttClient envia a mensagem ao servidor. Após envio da mensagem a conexão com o servidor é fechada, pelo método disconnect(). Como não foi definido o setRetained(), ou seja, se a mensagem será retida ou não no broker, o padrão é que não será retida ( setRetained(false) ). O QoS usado também será o padrão, ou seja, o número 1, na qual a mensagem é sempre

entregue pelo menos uma vez ( setQos(1) ).

O próximo passo é fazer, de fato, o sistema enviar a mensagem ao broker. Para a realização desta tarefa, na classe Main (classe criada com o WindowsBuilder) adicione a seguinte importação da Figura 34:

```
import java.awt.BorderLayout;  
import java.awt.EventQueue;  
import org.eclipse.paho.client.mqttv3.MqttException;
```

Figura 34: importação MqttException

Agora tem que alterar o evento de clique do botão, para quando este for clicado enviar a mensagem ao Broker. A Figura 35 exemplifica esta codificação.

```
JButton btnNewButton = new JButton("Enviar Soma");  
btnNewButton.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        //chamada para validar campos para aceitarem apenas número  
        if( ValidarCampoNumerico(textField) && ValidarCampoNumerico(textField_1) ) {  
            System.out.println(textField.getText() + " " +  
                               comboBox.getSelectedIndex() + " " +  
                               textField_1.getText());  
  
            //monta a string que será enviada para o broker  
            String mensagem = textField.getText() + " " +  
                               comboBox.getSelectedIndex() + " " +  
                               textField_1.getText();  
  
            //instancia o objeto para publicar as mensagens  
            Publisher publisher = new Publisher( mensagem );  
            //tratamento para envio de mensagem  
            try {  
                //chama método da classe Publiher para enviar a mensagem  
                publisher.enviarMensagemConta();  
            } catch (MqttException e) {  
                //imprime exceção no console  
                System.out.println(e);  
            }  
        }  
    }  
});
```

Figura 35: Evento de clique para envio de mensagem

## 7.5 Teste do envio da mensagem da aplicação para o Mosquitto Broker

Para testar o envio de mensagem, primeiramente execute o Eclipse Mosquitto Broker, abra o terminal e digite:

```
mosquitto_sub -h 127.0.0.1 -t enviar_conta
```

Na qual:

- **mosquitto\_sub**: é o Subscriber do broker mosquito
- **-h**: indica o ip do servidor broker
- **-t**: especifica o tópico que Subscriber do broker mosquito está assinando

Após executar o código do Mosquitto, abra o Eclipse e execute a aplicação, ainda com o broker ativo com o código especificado acima. Digite qualquer valor numérico nos dois campos e em seguida clique no botão "Enviar Conta". Verifique se a mensagem chegou corretamente no Broker, semelhante a Figura 36

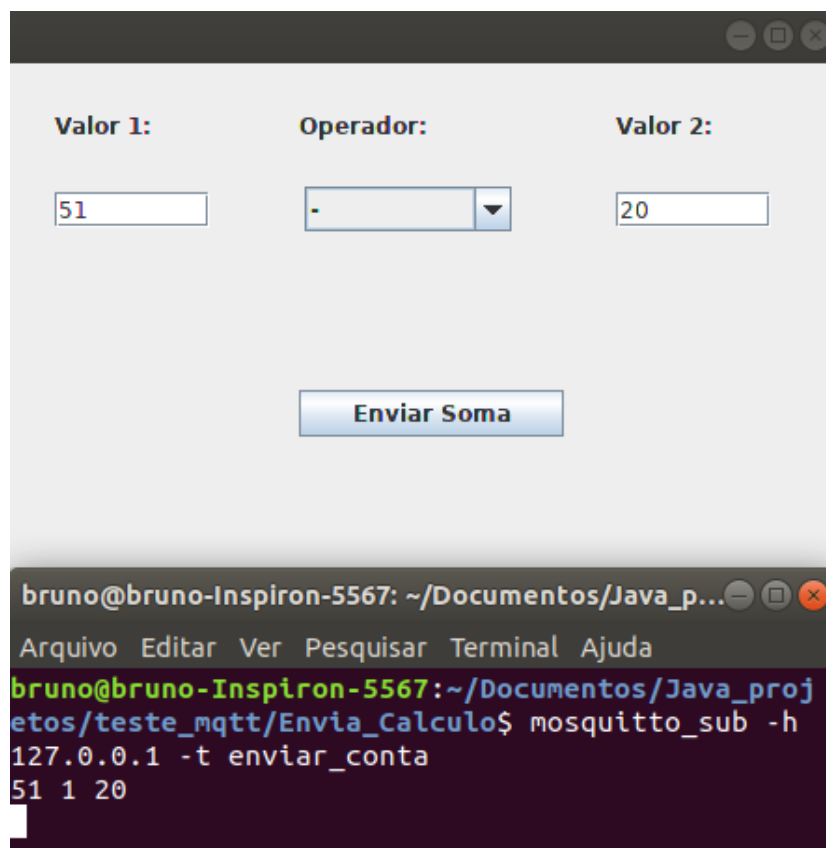


Figura 36: Teste de comunicação entre aplicação e servidor



## 8 Implementação do projeto Envia\_Resultado em JAVA

Para a implementação do projeto que receberá os valores, realizará o calculo e retornará a devida resposta, realize os passos das seguintes subseções, alterando o nome do projeto para Enviar\_Resultado.

- 7.1 Criação do projeto
- 7.2 Converter projeto para Maven Project

Após execução destes passos, crie a classe Main.java, classe responsável por iniciar a aplicação. Adicione as seguinte importações, referente ao MQTT, presente na Figura 37

```
import org.eclipse.paho.client.mqttv3.MqttClient;  
import org.eclipse.paho.client.mqttv3.MqttException;
```

Figura 37: Importações MQTT para recebimento de mensagens do Broker

Agora crie o seguinte método principal, na classe Main, para execução da aplicação, conforme Figura 38:

```
//Método principal  
public static void main(String[] args) throws MqttException {  
    //Local para onde será enviado a mensagem (broker)  
    MqttClient client=new MqttClient("tcp://localhost:1883", MqttClient.generateClientId());  
    /*Utilização da interface MqttCallback, que Permite que um aplicativo seja notificado  
    quando ocorrerem eventos assíncronos relacionados ao cliente*/  
    client.setCallback(new VerificaStatusServidor());  
    //Cria conexão com o servidor  
    client.connect();  
    //subscribe no tópico para receber os valores para realização da conta  
    client.subscribe("enviar_conta");  
}
```

Figura 38: Criação do método principal

O método setCallback() é o responsável por implementar uma interface na qual notifica a aplicação quando uma mensagem do broker chega. Esta interface será implementada através da classe VerificaStatusServidor. Para tanto, crie uma classe com este nome e realize as importações, de acordo com a Figura 39

```
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;  
import org.eclipse.paho.client.mqttv3.MqttCallback;  
import org.eclipse.paho.client.mqttv3.MqttException;  
import org.eclipse.paho.client.mqttv3.MqttMessage;
```

Figura 39: Importações para o MQTT

Após realização das devidas importações, o próximo passo é implementar a interface que avisa aplicação quando ocorre eventos assíncronos relacionados ao cliente. Para tanto, na linha de código no qual fica escrito o nome da classe adicione o seguinte código da Figura 40

Para testar se a aplicação está recebendo a mensagem, adicione o seguinte código na classe VerificaStatusServidor, conforme Figura 41

```
public class VerificaStatusServidor implements MqttCallback {
}
```

Figura 40: Interface MqttCallback

```
//método chamado quando cai conexão com o servidor
public void connectionLost(Throwable throwable) {
    System.out.println("Sem conexão com o servidor!");
}
//método invocado quando chega uma mensagem do broker
public void messageArrived(String s, MqttMessage mqttMessage) throws Exception {
    System.out.println("Mensagem:\t"+ new String(mqttMessage.getPayload()) );
}
//Chamado quando a entrega de uma mensagem foi concluída e todas as confirmações foram recebidas
public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
}
```

Figura 41: Código de teste

Execute as duas aplicações, e na aplicação Envia\_Resultado verifique se no console foi impresso a mensagem, semelhante ao exemplo da Figura 42

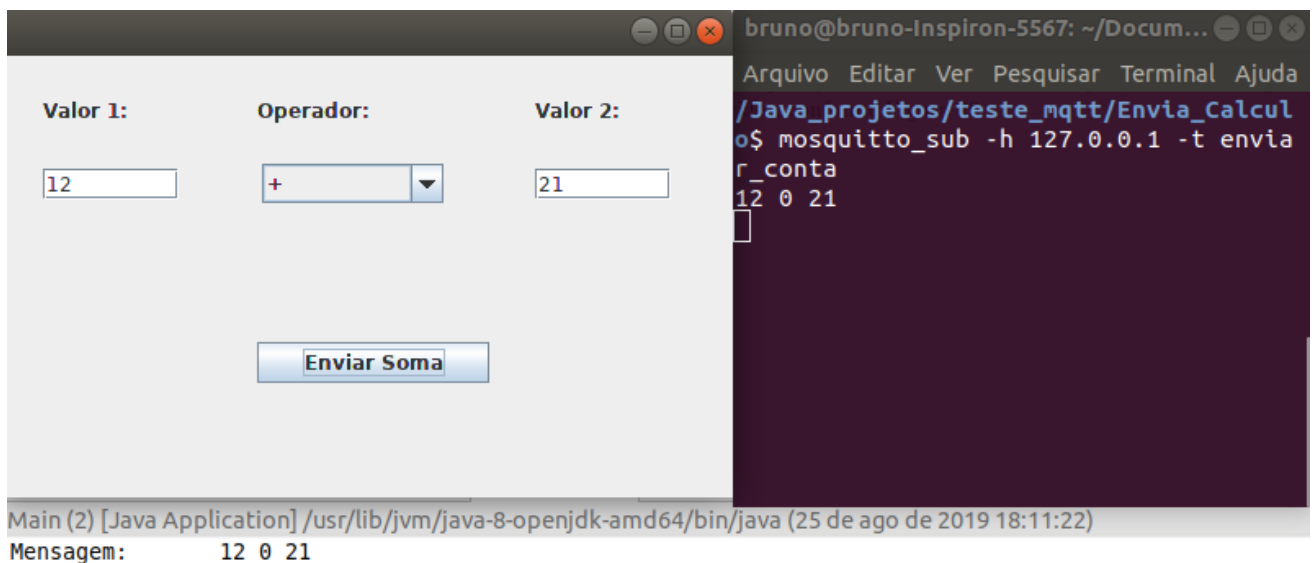


Figura 42: Primeiro teste de comunicação entre as aplicações

Agora que foi testado a conexão das duas aplicações será implementado a realização do cálculo dos dois valores e a apresentação dos resultados. No método `messageArrived()`, da classe `VerificaStatusServidor`, declare as seguintes variáveis da Figura 43

A linha `String[] textoSeparado = mensagem.split()` do código da Figura 43 separa a mensagem em um array, na qual a sua posição zero contém o primeiro valor. A posição um contém o operador de cálculo e a posição dois contém o segundo valor.

Agora que tem os valores e o operador, basta verificar qual o tipo de conta a se fazer. No caso foi utilizado um "switch - case", caso queira implementar mais funcionalidades de acordo com cada valor recebido...

A Figura 44 exhibe a implementação do "switch - case".

```
//método invocado quando chega uma mensagem do broker
public void messageArrived(String s, MqttMessage mqttMessage) throws Exception {
    //recebe a mensagem do broker
    String mensagem = new String(mqttMessage.getPayload());
    //divide a mensagem em um array
    String[] textoSeparado = mensagem.split(" ");
    //índice zero do array é o primeiro valor
    Double x = Double.parseDouble(textoSeparado[0]);
    //índice dois do array é o segundo valor
    Double y = Double.parseDouble(textoSeparado[2]);
    //índice um do array é o primeiro valor
    String op = textoSeparado[1];
    //variável que exibirá a mensagem de resultado
    String mensagemResultado = "";
    //resultado da conta
    Double result = null;
    //flag para indicar se tentou dividir com zero (para formatar a mensagem corretamente)
    int flag = 0;
```

Figura 43: Declaração de variáveis

```
//Verifica qual o tipo da conta a ser realizada
switch(op) {
    case "0":
        result = x + y;
        break;
    case "1":
        result = x - y;
        break;
    case "2":
        if(y == 0) {
            mensagemResultado = "Não existe divisão por zero!";
            flag = 1;
            break;
        } else {
            result = x / y;
        }
        break;
    case "3":
        result = x * y;
        break;
}
```

Figura 44: Switch - Case

O código responsável por exibir o resultado é apresentado na Figura 45

```
if(flag == 0) {
    mensagemResultado = String.format("%.2f", result);
}
//exibe resultado na tela
JOptionPane.showMessageDialog(null, mensagemResultado, "Resultado",
    JOptionPane.INFORMATION_MESSAGE);
```

Figura 45: Código para exibir mensagem

Por fim adicione as seguinte importações, referente ao Array que a mensagem recebida foi dividido e a tela de apresentação do resultado, conforme Figura 46

```
import java.util.Arrays;
import javax.swing.JOptionPane;
```

Figura 46: Importação JOptionPane e Arrays

## 8.1 Limpar cache do Mosquitto Broker

Antes da execução das duas aplicações é recomendado que se limpe o cache do broker. Para isto basta executar o seguinte comando no terminal:

```
mosquitto_pub -h 127.0.0.1 -t enviar_conta -n -r -d
```

Na qual:

- **-h** endereço do servidor
- **-t** tópico das mensagens

## 8.2 Enviar resposta para o sistema Envia\_Calculo

Para que seja possível enviar a resposta para o sistema Enviar\_Calculo, primeiramente crie uma nova, com o nome PublisherEnviarResultado e adicione as seguintes importações, conforme exemplo da Figura 47

```
package enviar_calculo;

//Imports para utilizar o protocolo MQTT
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
```

Figura 47: Importação para MQTT Envia\_Resultado

Após importações para utilizar o MQTT, crie o seguinte construtor, para receber a String do retorno do resultado da operação realizada. A Figura 48 exibe o construtor desta classe.

```
//String que enviará a conta para o outro sistema
String mensagemEnviarResultado;
//construtor da classe para enviar a mensagem
public PublisherEnviarResultado(String mensagemEnviarResultado){
    this.mensagemEnviarResultado = mensagemEnviarResultado;
}
```

Figura 48: Construtor da classe PublisherEnviarResultado

Por fim crie o método enviarMensagemResultado(), que é bem semelhante ao método enviarMensagemConta(), apresentado em 7.4. A Figura 49 apresenta este método.

A única diferença para este método apresentado na Figura 49 é que foram definidos o QoS e a política de reter mensagem. Referente aquela, a mensagem é sempre entregue exatamente uma vez e em relação a última a mensagem mais recente ficará retida no broker. Os métodos responsáveis por isto são setQos() e setRetained() respectivamente.

Para enviar a mensagem com o resultado, para o sistema Envia\_Calculo, basta adicionar o trecho de código da Figura 50 antes de fechar a chave do método messageArrived() da classe VerificaStatusServidor.

```

//Método para enviar a mensagem para o mosquitto
public void enviarMensagemResultado() throws MqttException {
    //Local para onde será enviado a mensagem (broker)
    MqttClient client = new MqttClient("tcp://localhost:1883", MqttClient.generateClientId());
    //abre conexão com o broker (Mosquitto)
    client.connect();
    //objeto de envio de mensagem do broker
    MqttMessage mensagem = new MqttMessage();
    //Payload, conteúdo da mensagem montagem da mensagem que será enviada ao broker
    mensagem.setPayload(mensagemEnviaResultado.getBytes());
    //QoS - A mensagem é sempre entregue exatamente uma vez.
    mensagem.setQos(2);
    //Mensagem não ficará retida no Broker (ultima mensagem não fica retida)
    mensagem.setRetained(false);
    //Publica a mensagem, com seu tópico (para alguém se escrever neste tópico) e a mensagem montada
    client.publish("enviar_resultado", mensagem);
    //fecha conexão com broker mosquitto
    client.disconnect();
}

```

Figura 49: Método enviarMensagemResultado

```

//Instancia classe para enviar o resultado para Envia_Calculo
PublisherEnviarResultado publisherEnviarResultado =
    new PublisherEnviarResultado(mensagemResultado);

//tratamento para envio de mensagem
try {
    //chama método da classe PublisherEnviarResultado para enviar a mensagem
    publisherEnviarResultado.enviarMensagemResultado();
} catch (MqttException e) {
    //imprime exceção no console
    System.out.println(e);
}

```

Figura 50: Enviar Resultado

### 8.3 Testando código

Abra o terminal do seu Sistema Operacional e limpe o cache do Mosquitto, por exemplo:

```
mosquitto.pub -h 127.0.0.1 -t enviar_resultado -n -r -d
```

Atente-se para o nome do tópico que está sendo limpo o cache. Após realização desta tarefa, ainda no terminal do Mosquitto, execute o seguinte comando para subscribe no tópico de recebimento da resposta:

```
mosquitto.sub -h 127.0.0.1 -t enviar_resultado
```

Execute as duas aplicações e envie quaisquer valores e operando para observarmos o resultado, conforme exemplo das Figuras 51 e 52

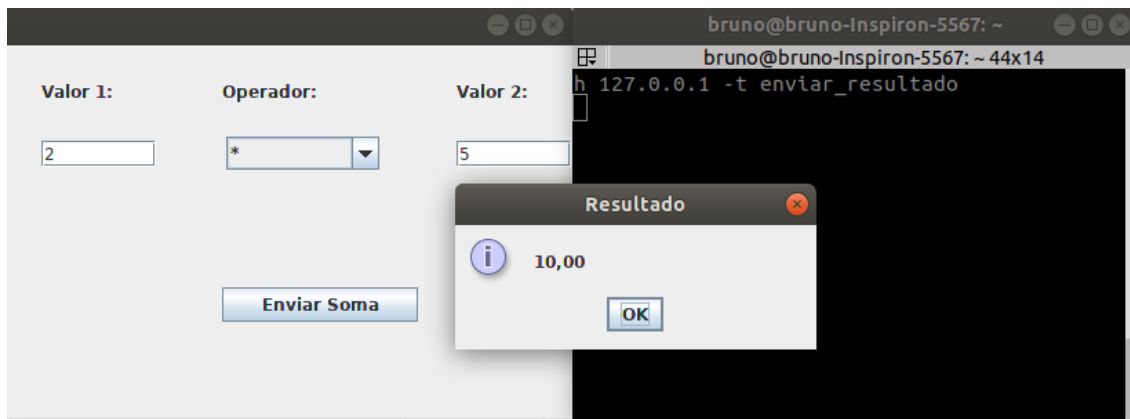


Figura 51: Teste Enviar Resultado

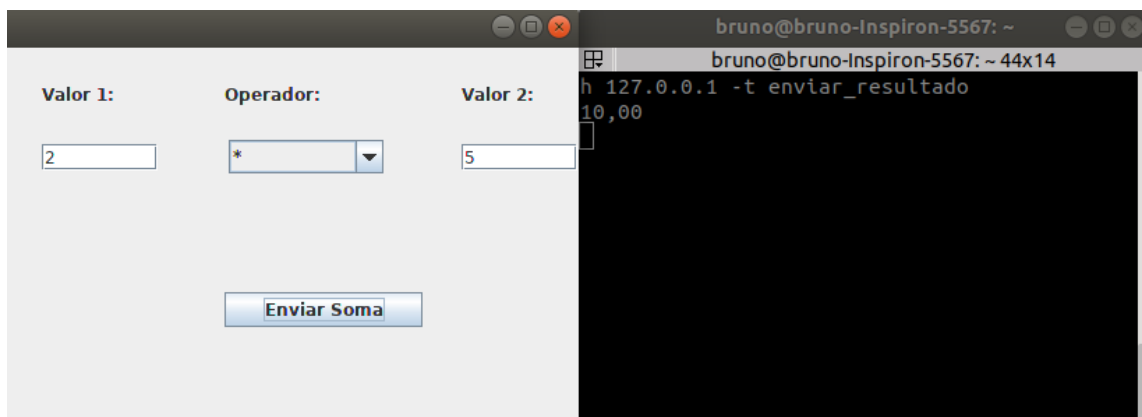


Figura 52: Teste Enviar Resultado Mosquitto

Perceba que o resultado só será exibido no Mosquitto (Figura 52) após clicar em OK, da janela "Resultado", apresentado na Figura 51.

O próximo passo é implementar o recebimento da mensagem na aplicação Envia\_Calculo.

## 9 Implementação da mensagem de retorno no projeto Envia\_Calculo

O primeiro passo será criar a classe Subscriber, para receber a mensagem de retorno do cálculo. Após a classe ser criada adicione as seguinte importações na classe, conforme Figura 53

```
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
```

Figura 53: Importações para o MQTT

Em seguida implemente a interface MqttCallback e os métodos obrigatórios, conforme Figura 54

Configure o método messageArrived(), que identifica que uma mensagem chegou do broker, para exibir o resultado da operação através de um JOptionPane, semelhante a Figura 55



```

public class Subscriber implements MqttCallback {
    //método chamado quando cai conexão com o servidor
    public void connectionLost(Throwable throwable) {
        System.out.println("Sem conexão com o servidor!");
    }
    //método invocado quando chega uma mensagem do broker
    public void messageArrived(String s, MqttMessage mqttMessage) throws Exception {
    }
    //Chamado quando a entrega de uma mensagem foi concluída e todas as
    //confirmações foram recebidas
    public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
    }
}

```

Figura 54: Métodos obrigatórios e interface

```

//método invocado quando chega uma mensagem do broker
public void messageArrived(String s, MqttMessage mqttMessage) throws Exception {
    //recebe a mensagem do broker
    String mensagem = new String(mqttMessage.getPayload());

    //exibe resultado na tela
    JOptionPane.showMessageDialog(null, mensagem, "Resultado Recebido",
        JOptionPane.INFORMATION_MESSAGE);
}

```

Figura 55: Métodos para exibir resultado

Agora precisamos implementar o callback na main do Envia\_Calculo para obter mensagem de retorno do broker. Abra classe Main e adicione o seguinte código no método main, para receber o resultado do calculo, pela Figura 56

```

public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Main frame = new Main();
                frame.setVisible(true);
                //Local para onde será enviado a mensagem (broker)
                MqttClient client=new MqttClient("tcp://localhost:1883",
                    MqttClient.generateClientId());
                /*Utilização da interface MqttCallback, que Permite que um
                 * aplicativo seja notificado
                 * quando ocorrerem eventos assíncronos relacionados ao cliente*/
                client.setCallback(new Subscriber());
                //Cria conexão com o servidor
                client.connect();
                //subscribe no tópico para receber os valores para realização da conta
                client.subscribe("enviar_resultado");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

```

Figura 56: Receber mensagem

## 10 Criando arquivo de configuração para o IP do servidor

Para cada aplicação, você terá que criar uma classe que conterá o endereço IP do servidor broker. Para isto crie uma nova classe chamada ConfigServer nas duas aplicações e adicione o código da Figura 57 na nova classe criada.

```
public class ConfigServer {
    String enderecoServidor = "tcp://localhost:1883";
    //recupera endereco ip
    public String getEnderecoServidor() {
        return enderecoServidor;
    }
    //altera endereco ip
    public void setEnderecoServidor(String enderecoServidor) {
        this.enderecoServidor = enderecoServidor;
    }
}
```

Figura 57: Classe para configurar ip do servidor

A variável "enderecoServidor" receberá o ip da máquina onde o broker está instalado. Após criar esta mesma classe nos projetos Envia\_Calculo e Envia\_Resultado teremos que realizar algumas alterações para poder utilizar esta classe.

Na aplicação Envia\_Calculo, abra a classe **Main.java**. Na instanciação da classe MqttClient, altere esta linha da forma apresentada na Figura 58:

```
ConfigServer endIp = new ConfigServer();
MqttClient client=new MqttClient(endIp.getEnderecoServidor(),
    MqttClient.generateClientId());
```

Figura 58: Alteração para atribuir IP do servidor broker

Perceba que a classe ConfigServer foi instanciada e no construtor da instanciação da classe MqttCliente foi chamado o método getEnderecoServidor(), que informa a máquina onde o broker está instalado. Realize o mesmo processo de codificação no arquivo **Publisher.java**.

Na aplicação Envia\_Resultado realize as mesmas alterações para as classes **Main.java** e na classe **PublisherEnviarResultado.java**.

## 11 Teste Completo do sistema

Será apresentado nesta seção como executar a comunicação entre duas máquinas distintas

### 11.1 Limpando cache do Broker

Para testar completamente o sistema primeiramente deve limpar o cache do Mosquitto:

```
mosquitto_pub -h 127.0.0.1 -t enviar_resultado -n -r -d
mosquitto_pub -h 127.0.0.1 -t enviar_conta -n -r -d
```

Em seguida devemos saber o IP do servidor, ou seja, da máquina no qual o Mosquitto Broker está instalado.



## 11.2 Descobrir IP do Broker no Linux

Abrir o terminal (pressione Ctrl + Alt + T). Na janela aberta do terminal digite o seguinte comando:

**ifconfig**

E em seguida aperte "Enter". O IP da máquina será exibido semelhante ao exemplo da Figura 59:

```
wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
  inet 192.168.0.106 netmask 255.255.255.0 broadcast 192.168.0.255
  inet6 fe80::c5f6:9bbb:ec84:dea1 prefixlen 64 scopeid 0x20<link>
  ether e8:9e:b4:65:24:39 txqueuelen 1000 (Ethernet)
  RX packets 24474 bytes 26797921 (26.7 MB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 16639 bytes 2533137 (2.5 MB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

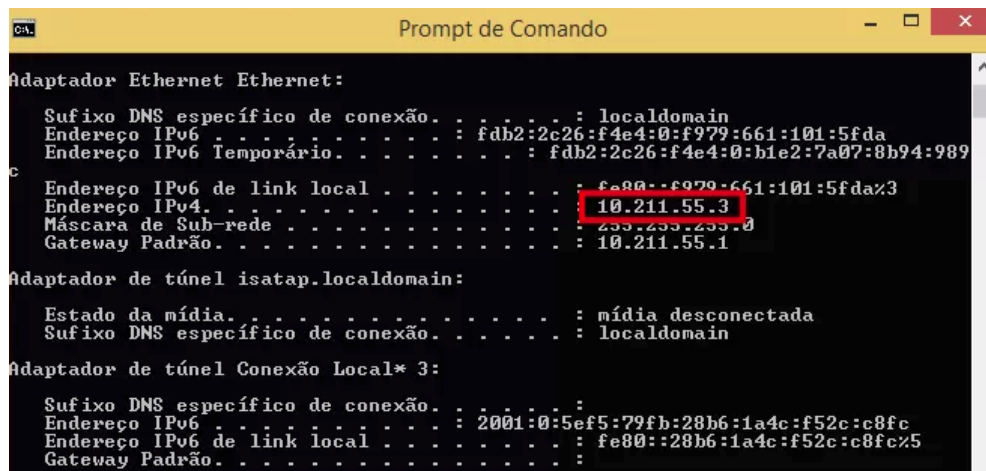
Figura 59: Descobrir IP no Linux

## 11.3 Descobrir IP do Broker no Windows

Abra o terminal do Windows (clique em iniciar e digite CMD). Na janela que abrir digite o seguinte comando:

**ipconfig**

O IP será exibido conforme Figura 60:



```

Adaptador Ethernet Ethernet:
    Sufixo DNS específico de conexão. . . . . : localdomain
    Endereço IPv6 . . . . . : fdb2:2c26:f4e4:0:f979:661:101:5fda
    Endereço IPv6 Temporário. . . . . : fdb2:2c26:f4e4:0:b1e2:7a07:8b94:989
    c
    Endereço IPv6 de link local . . . . . : fe80::f979:661:101:5fda%3
    Endereço IPv4. . . . . : 10.211.55.3
    Máscara de Sub-rede . . . . . : 255.255.255.0
    Gateway Padrão. . . . . : 10.211.55.1

Adaptador de túnel isatap.localdomain:
    Estado da mídia. . . . . : mídia desconectada
    Sufixo DNS específico de conexão. . . . . : localdomain

Adaptador de túnel Conexão Local* 3:
    Sufixo DNS específico de conexão. . . . . :
    Endereço IPv6 . . . . . : 2001:0:5ef5:79fb:28b6:1a4c:f52c:c8fc
    Endereço IPv6 de link local . . . . . : fe80::28b6:1a4c:f52c:c8fc%5
    Gateway Padrão. . . . . :
```

Figura 60: Descobrir IP no Windows

## 11.4 Estabelecendo comunicação entre as máquinas

Após a descoberta do IP do Broker, basta alterar "endereçoServidor" da classe **ConfigServer.java** das duas aplicações para o endereço do Broker. O endereço fornecido para esta variável deve seguir o seguinte padrão:

**tcp://endereço\_ip:1883**

Conforme exemplo da Figura 61

```
String enderecoServidor = "tcp://192.168.0.106:1883";
```

Figura 61: Atribuição endereço IP nas aplicações

## 12 Possíveis soluções para erro de comunicação

Caso as máquinas não estejam comunicando, verifique os seguintes procedimentos:

- Verifique se a porta 1883 está disponível;
- Verifique se o antivírus está impedindo a comunicação;
- Verifique se digitou a IP correta do servidor;
- Verifique se o firewall do sistema operacional não está bloqueando;
- Verifique se as máquinas na rede estão com a mesma faixa de IP, se estão de fato na mesma rede

## 13 Código fonte das duas aplicações no Github

Para realizar o download do projeto Envia\_Conta acesse o link:

<https://github.com/BrunoLunardi/Java-MQTT-Envia-Conta>

E do projeto Envia\_Resultado o link do Github é:

<https://github.com/BrunoLunardi/Java-MQTT-Envia-Resultado.git>

## 14 Conclusão

Com o desenvolvimento do projeto foi possível a implementação de duas aplicações que utilizam o protocolo de comunicação MQTT. Nesta comunicação, as aplicações foram escritas em Java, e executavam as funções de clientes do tipo publish, subscribe e/ou publish/subscribe. As aplicações desenvolvidas obtiveram resultados satisfatórios, uma vez que os resultados esperados foram atingidos, no que se diz respeito ao funcionamento do broker, funcionamento da comunicação via MQTT e fluxo de informações entre clientes MQTT através de um servidor.

No desenvolvimento, pode-se observar o quão simples é a implementação do protocolo MQTT e notar a variedade de aplicações que podem ser construídas com base neste protocolo, principalmente, aquelas que trabalham com muitas informações curtas, as que necessitam da possibilidade de comunicação bilateral ou as que precisam de conectar várias máquinas e sensores ao mesmo tempo.

Podemos também ressaltar como vantagens no uso do MQTT: a codificação simples, que possibilita que o protocolo seja executado mesmo em sistemas poucos modernos ou com falhas de armazenamento; a não sobrecarga do sistema, tendo em vista que apenas o necessário passa através dele; o domínio público, o que o torna mais flexível e que possa ser instalado em qualquer rede ou hardware e, a segurança, pois ele consegue usar criptografia, além de toda a questão de QoS.

Ademais, pela facilidade de implementação e por trabalhar com informações mais leves e pequenas, ele se mostra uma excelente opção dentro do mercado de IoT, já que não será necessário muito banda nem muito processamento para conseguir estabelecer uma conexão. Isso também faz com ele tenha um bom desempenho em situações que em que é necessário baixo custo e onde as conexões são precárias.

## Referências

- [Andrei B. B. Torres, 2016] Andrei B. B. Torres, Atslands R. Rocha, J. N. d. S. (2016). Analise de desempenho de brokers mqtt em sistema de baixo custo. [Online; acessado em 27 de agosto].
- [IBM, 2018a] IBM (2018a). Persistência de mensagem em clientes mqtt. [Online; acessado em 25 de agosto].
- [IBM, 2018b] IBM (2018b). Qualidades de serviço fornecidas por um cliente mqtt. [Online; acessado em 25 de agosto].
- [Maven, 2019] Maven (2019). Apache maven project. [Online; acessado em 29 de agosto].
- [Mosca, 2016] Mosca (2016). Mosca lib/server.js. [Online; acessado em 26 de agosto].
- [Mosquitto, 2019] Mosquitto, E. (2019). Eclipse mosquitto<sup>TM</sup>. [Online; acessado em 28 de agosto].
- [Team, 2015] Team, T. H. (2015). Conhecendo o mqtt. [Online; acessado em 25 de agosto].
- [Team, 2019] Team, T. H. (2019). Understanding the mqtt protocol packet structure. [Online; acessado em 22 de agosto].
- [Yuan, 2017] Yuan, M. (2017). Conhecendo o mqtt. [Online; acessado em 22 de agosto].