A- Comunicação Síncrona

- estilo cliente-servidor com pedido/resposta convencional
- tecnologias: REST, GraphQL, Thrift

B- Comunicação Assíncrona

- a resposta pode acontecer num momento posterior;
  - a) é possível com variantes do modelo cliente-servidor, com um endpoint do lado de quem faz o pedido, para execução de uma callback em momento diferido
  - b) mas usualmente assenta num sistema de filas de mensagens, ou publish/subscribe
- uma resposta pode ser encaminhada para múltiplos destinatários;
- pode haver intermediários

Publish/subscribe e comunicação baseada em filas de mensagens

- o processo que lê não tem de saber tudo sobre a origem dos dados (o processo que gera os dados)
- facilita o atendimento de um nº crescente de leitores/recetores
- facilita alguma tolerância a falhas do lado dos geradores de mensagens, dado que as filas continuam a dar resposta aos clientes
- abordagem em linha com os microsserviços (ver *messaging & microservices*)
- potenciais vantagens em
  - escala e alta disponibilidade
  - possibilidade de agregar funcionalidades como filtros e monitorização em larga escala
  - facilidade de gestão de diferentes ritmos: geração de mensagens *vs* a capacidade de ler/receber mensagens por parte de clientes e ritmos distintos

Protocolos para comunicação assíncrona

- AMQP: *Advanced Message Queuing Protocol* (AMQP) é uma norma aberta; protocolo ao nível de aplicação, binário, para *message-oriented middleware*
- STOMP: *Simple Text Oriented Messaging Protocol* (STOMP), protocolo para troca de dados em formato textual, sobre HTTP
- MQTT: *Message Queue Telemetry Transport* (MQTT) é uma norma para *publish/subscribe based lightweight messaging* muito usada em IoT (transmissão desde sensores e outros equipamentos terminais de baixa capacidade de processamento e possível ligação intermitente)
  - https://mqtt.org/ - ver "*MQTT Publish / Subscribe Architecture*"

Broker & Event processing

Um **broker**, ou módulo gestor de filas de mensagens, é usado na comunicação entre sistemas ou processos. Estas filas permitem que emissores e recetores comuniquem com flexibilidade, e em momentos diferentes. As mensagens podem incluir pedidos e respostas convencionais, alertas, ou registos de eventos.

O processamento de eventos (*event processing*) é o tratamento ou execução de operações em função dos eventos reportados ou observados diretamente. (leitura complementar)

Public cloud-based event processing

- Google Pub/Sub - (ver conceitos e fluxo)
- MS Azure Event Hubs

- Amazon Kinesis

Message brokers populares

- RabbitMQ
  - *message broker* de uso genérico com suporte para protocolos MQTT, STOMP e AMQP
  - pode envolver cluster de nós ou mesmo uma federação de clusters
  - pode incluir comunicação síncrona e assíncrona
- Apache Kafka
  - *open-source distributed event streaming platform*
  - pode usar-se como *message broker*
- Apache ActiveMQ
  - serviço open-source de mensagens, baseado em Java
  - suporte para MQTT, STOMP e AMQP
  - rede de *brokers/ clustering* para distribuição de carga
  - https://activemq.apache.org/

Sistema a comunicar com MQTT, em Java

O que é preciso?

- ter um Broker ativo
- biblioteca que suporte o protocolo, como Apache Paho (ver API aqui, especialmente a classe MqttClient)
  - implementar o emissor (aplicação cliente que gera mensagens, que publica)
  - implementar o recetor (aplicação cliente que subscreve)

Leitura por alto:
aqui

Opte uma destas opções:

1.a (**use esta opção da primeira vez**)- Obter o projeto com a estrutura base, aqui;

1.b (**alternativa a 1.a**)- Criar um novo projeto Java com Maven, incluindo a biblioteca Apache Paho. No pom.xml deve ter:

```
<repositories>
    <repository>
        <id>Eclipse Paho Repo</id>
        <url>https://repo.eclipse.org/content/repositories/paho-releases/</url>;
    </repository>
</repositories>
<dependencies>
    <dependency>
        <groupId>org.eclipse.paho</groupId>
        <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
        <version>1.2.1</version>
    </dependency>
</dependencies>
```

Publisher Client

No package sd.mqtt, crie uma nova classe com o nome Publisher.

No método main(),
aplique o essencial para uma experiência de escrita:

```
String clientId     = "JavaSample";
String broker= "tcp://broker.mqttdashboard.com:1883";  // URL de um Broker ativo
String topic        = "MQTT Examples";
String content      = "Message from MqttPublishSample em aula de SDist"; // mensagem a enviar
int qos             = 2;

MqttClient sampleClient = new MqttClient(broker, clientId, new MemoryPersistence());
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setCleanSession(true);
```

```
System.out.println("Connecting to broker: "+broker);
sampleClient.connect(connOpts);
```

```
System.out.println("Connected");
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(qos);

sampleClient.publish(topic, message);
System.out.println("Message published");

sampleClient.disconnect();
sampleClient.close();
```

Resolva o necessário com o código (import...)... e execute a aplicação. Espera-se algo como:

```
-------------------------< sd:mqtt.client1 >--------------------------
Building mqtt.client1.inicial 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ mqtt.client1 ---
Connecting to broker: tcp://broker.mqttdashboard.com:1883
Connected
Message published
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

E se quiser executar na linha de comandos? Por exemplo assim:

```
$ mvn exec:java -Dexec.mainClass=sd.mqtt.Publisher
```

Subscriber Client

Vamos agora tratar da aplicação para receber mensagens.
No mesmo package, crie uma nova classe Subscriber, que inclua:

```
String clientId     = "myClient2";
String topic        = "MQTT Examples";                  // confirme que é o tópico certo
String broker= "tcp://broker.mqttdashboard.com:1883";   // verifique se é o mesmo endereço
MqttClient sampleClient = new MqttClient(broker, clientId, new MemoryPersistence());
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setCleanSession(true);
```

```
        sampleClient.setCallback(new MqttCallback() {
            public void connectionLost(Throwable cause) { //Called when the client lost the connection to the broker
            }
            public void connectComplete(boolean reconnect, java.lang.String serverURI) {
            }
            public void messageArrived(String topic, MqttMessage message) throws Exception {   // método acionado ao receber
uma msg
                System.out.println("messageArrived:");
                System.out.println(topic + ": " + new String(message.getPayload()) );
            }
            public void deliveryComplete(IMqttDeliveryToken token) {//Called when a outgoing publish is complete
            }
        });

        System.out.println("Connecting to broker: "+broker);
        sampleClient.connect(connOpts);
        System.out.println("Connected");
        sampleClient.subscribe(topic);
        /*
        for (int i=1; i<30; i++) {
            System.out.println("\t ... "+i);
            Thread.sleep(1000);
        }
        */
        sampleClient.disconnect();
        sampleClient.close()
```

Resolva o necessário com o código (import...)... e execute a aplicação.

Descomente o ciclo e volte a executar a aplicação... e entretanto execute o Publisher...

Documentação: tópicos e *retained messages*

Estude na API:

- nível de QoS (semântica para o serviço),
- sintaxe para organização dos tópicos
- e retenção de mensagens em tópicos (*retained*)

Sobre a retenção de mensagens, faça experiências:

- Tentar colocar uma nova "retained message"; ligar um novo subscritor e ver se recebe a mensagem.
- Tentar ligar o subscritor num novo tópico... e ver se recebe imediatamente alguma mensagem.
- E se definirmos também uma retained message para esse tópico... e voltar a ativar os subscritores?

Para além do MQTT Broker indicado no código acima, poderia usar qualquer outro.
O HiveMQ disponibiliza um broker de testes que também pode usar:

```
        Broker: broker.hivemq.com
        TCP Port: 1883
```

MQTT Broker "local"

Para ter todo o sistema sob controlo, pode ainda ativar e controlar o próprio MQTT Broker.
Algumas opções são:

- Cassandana
- ActiveMQ

Cassandana

"an open source MQTT message broker which is entirely written in Java".

Download:
https://github.com/mtsoleimani/cassandana

```
mvn package
```

Se não quiser instalar no seu sistema, pode executar diretamente:

```
$ java -jar target/cassandana-jar-with-dependencies.jar
```

Teste a comunicação dos seus clientes MQTT com este broker, direcionando-os para este broker (""tcp://localhost:1883"").

Apache ActiveMQ

https://activemq.apache.org/components/classic/

Download

Executar:

- apache-activemq-5.16.2$ ./bin/activemq console

Para parar

- Ctrl+C
- ou ./bin/activemq stop

Antes de correr os programas cliente, aceda à interface de administração do broker:

http://127.0.0.1:8161/admin/
admin / admin

Veja os tópicos, os subscribers e producers.

Execute as aplicações anteriores, direcionando-as para este broker (""tcp://localhost:1883"").

Volte a consultar a informação existente sobre tópicos e a lista de subscribers conhecidos.

Leia sobre a Client Will Message:

- https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/

Transmissão assíncrona

Veja na API opções alternativas para o cliente enviar as mensagens para o broker...

Em particular na execução assíncrona de publish(), como saber se a entrega foi concretizada (via token, ou via callback).
Note que o connect() de MqttAsyncClient fica a executar em segundo plano, pelo que poderia tentar o publish() antes mesmo de estar ligado. Deve confirmar que o procedimento de connect está concluído (com o IMqttToken devolvido pelo método connect()).

MQTT Client em Python

https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php

Apontadores

- https://www.eclipse.org/paho/
- https://pypi.org/project/paho-mqtt/
- https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/